NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

IMPLEMENTATION OF A CONFIGURABLE FAULT TOLERANT PROCESSOR (CFTP)

by

Steven A. Johnson

March 2003

Thesis Advisor:	Herschel H. Loomis
Second Reader:	Alan A. Ross

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** March 2003	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: Implementation of a Configurable Fault Tolerant Processor (CFTP)	**5. FUNDING NUMBERS**
6. AUTHOR(S) Johnson, Steven A.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(maximum 200 words)*

The space environment has unique hazards that force electronic systems designers to use different techniques to build their systems. Radiation can cause Single Event Upsets (SEUs) which can cause state changes in satellite systems. Mitigation techniques have been developed to either prevent or recover from these upsets when they occur.

At the same time, modifying on-orbit systems is difficult in a hardwired electronic system. Finding an alternative to either working around a mistake or having to keep the same generation of technology for years is important to the space community. Newer programmable logic devices such as Field Programmable Gate Arrays (FPGAs) allow for emulation of complex logic circuits, such as microprocessors. FPGAs can be repro-grammed as necessary, to account for errors in design, or upgrades in software logic circuits.

In an effort to provide one solution for both of these issues, this research was undertaken. The Configurable Fault Tolerant Processor (CFTP) emulates three identical processors, using Triple Modular Redundancy (TMR) to mitigate SEUs on a radiation tolerant FPGA. With the reconfigurable capabilities of FPGA technology, as newer processors can be emulated, these new configurations can be uploaded to the satellite as software code, thereby actually upgrading the processor in flight. This research used a 16-bit Reduced Instruction Set Computer (RISC) processor as its cores. This thesis describes how the Harvard architecture of the processor is interfaced with the Von Neumann architecture of the memory. It also develops the process by which errors are detected and corrected, as well as recorded. The end result is a design simulation ready for implementation on an FPGA.

14. SUBJECT TERMS Fault Tolerant Computing, Triple Modular Redundancy (TMR), Field Programmable Gate Array (FPGA), Single Event Upset (SEU), 16-Bit RISC	**15. NUMBER OF PAGES** 139
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

THIS PAGE INTENTIONALLY LEFT BLANK

IMPLEMENTATION OF A CONFIGURABLE FAULT TOLERANT PROCESSOR (CFTP)

Steven A. Johnson
Lieutenant, United States Navy
B.S., United States Naval Academy, 1996

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 2003

Author:	Steven A. Johnson
Approved by:	Herschel H. Loomis Thesis Advisor
	Alan A. Ross Second Reader
	John P. Powers Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The space environment has unique hazards that force electronic systems designers to use different techniques to build their systems. Radiation can cause Single Event Upsets (SEUs) which can cause state changes in satellite systems. Mitigation techniques have been developed to either prevent or recover from these upsets when they occur.

At the same time, modifying on-orbit systems is difficult in a hardwired electronic system. Finding an alternative to either working around a mistake or having to keep the same generation of technology for years is important to the space community. Newer programmable logic devices such as Field Programmable Gate Arrays (FPGAs) allow for emulation of complex logic circuits, such as microprocessors. FPGAs can be reprogrammed as necessary, to account for errors in design, or upgrades in software logic circuits.

In an effort to provide one solution for both of these issues, this research was undertaken. The Configurable Fault Tolerant Processor (CFTP) emulates three identical processors, using Triple Modular Redundancy (TMR) to mitigate SEUs on a radiation tolerant FPGA. With the reconfigurable capabilities of FPGA technology, as newer processors can be emulated, these new configurations can be uploaded to the satellite as software code, thereby actually upgrading the processor in flight. This research used a 16-bit Reduced Instruction Set Computer (RISC) processor as its cores. This thesis describes how the Harvard architecture of the processor is interfaced with the Von Neumann architecture of the memory. It also develops the process by which errors are detected and corrected, as well as recorded. The end result is a design simulation ready for implementation on an FPGA.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank all the professors, technicians, and students of the Naval Postgraduate School who made this research possible. Many of them may not realize the magnitude of their efforts, but the author does.

Special thanks are owed to these individuals for their assistance:

To Drs. Loomis and Ross for your infinite knowledge and especially patience with me throughout the research.

To Dr. Clark for both your processor and assistance in understanding how the KDLX processor worked.

To David Rigmaiden for entertaining even the most bizarre of questions on how to operate the Xilinx tools.

To Major Dean Ebert for your collaboration in the SERB process, patience with my computer illiteracy, and your good humor throughout.

To Lieutenant Paula Travis of the Space Test Program Office, for your invaluable assistance in getting the CFTP up to speed in the SERB process.

To Damon Van Buren and Tilan Langley of SEAKR Engineering, you were wonderful resources for FPGA and the space environment.

To Captain Charles Hulme and First Lieutenant Yuan Rong for taking over the SERB process with aplomb and skill.

To the Bearded Devil, who kept me motiviated the last quarter.

And most especially to my loving wife for her patience and perseverance in the long hours I was in the lab.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

The harsh environment of space creates unique problems for hardware used in space versus equivalent ground based systems. Space-borne systems face susceptibility to errors induced by particles passing through the microchips without sufficient protection from radiation. There are many effects that can cause errors, including total dose tolerance, Single Event Latchup (SEL) and Single Event Upset (SEU). This research will focus on mitigating SEUs. It is assumed that total dose and SEL effects are accounted for by other means.

Before the technology boom of the 1990s, purchasing radiation hardened equipment was not difficult. But as the market for non-hardened systems became significantly larger, and the demand for radiation hardened systems remained the same, companies offering the hardened devices either increased the price and time required to deliver, or stopped offering devices altogether.

This change in the market dynamics greatly affected how satellite systems function. With such a long lead time for processors to be ready for flight, older and less capable designs were being flown, and costing more than in the past when the satellite designers could expect a close to high end processor.

For example, current satellite processors are typically equivalent to a 386 generation microprocessor. The average desktop computer found in most homes today is a Pentium III[1]. Additionally, the cost for purchasing the radiation hardened 386 is much higher than the entire desktop computer.

New techniques are required to accommodate the continued problems encountered by radiation while simultaneously addressing the newer problems related to development time. Many of the current designs for space systems include the use of software redundancy, examples of which can be found in Table 1. In addition, the use of reprogrammable devices is now very common for satellite design. But little research has gone into combining all of this new technology, mostly due to the cost of satellites.

[1] Pentium is a registered trademark of Intel Corporation.

The most promising options for radiation mitigation include more flexible software solutions implemented on radiation tolerant, not hardened, microchips. One such option is Triple Modular Redundancy (TMR).

In TMR, the system is replicated three times, and the outputs of the system are voted, with the majority vote becoming the overall system output. In the specific case of the CFTP, three microprocessors are operating in lock-step. The output of the processors is the input to the voting circuitry. The outputs of the voters is the overall system output.

At this point the system being designed has a reliable way to circumvent single event errors without resorting to radiation hardening techniques. But the problems of rapid development and reconfigurability still must be addressed.

Reprogrammable devices satisfy both rapid development and reconfigurability. The most common device is the Field Programmable Gate Array (FPGA). An FPGA is a set of generic logic devices that can be programmed and reprogrammed a large number of times "in the field," meaning by the end user, not just the manufacturer. Any circuit can be emulated on the FPGA, assuming the FPGA has a sufficient amount of logic blocks to mimic the desired device.

FPGAs are currently used on many space platforms, but not primarily for their on-orbit reconfigurable qualities. Most of them are used as Application Specific Integrated Circuits (ASICs). Since ASICs cannot be reconfigured once they are built, correcting mistakes can become expensive for designers. But an FPGA design can be changed at any point in time, and the FPGA reprogrammed, up to the launch. On-orbit reconfiguration is a natural progression from current usage.

Based on previous NPS research, Xilinx Virtex family FPGAs were chosen for this research. Xilinx Virtex FPGAs are total dose tolerant up to 100 kilorads and are also latchup immune. For the radiation environments of this application, 100 kilorads is enough tolerance to allow its 1 year minimum flight duration.

The intent of this research is to develop a single system-on-a-chip design for a Configurable Fault Tolerant Processor (CFTP). It includes the design of the processor, the voter, error interrupt designs, memory controller, and error syndrome storage.

The processor used for this research is a KDLX 16-bit Reduced Instruction Set Computer (RISC) [2]. The processor was created by previous NPS research. Each processor will run the same set of instructions and the outputs will be voted. The processors and voters are in the component names TMR.

The processors have a Harvard architecture and memory is Von Neuman in design. A memory controller, called the *Reconciler*, was designed to control the requirements of the processor. The processors run at a clock rate which is half of the remaining devices clock rate. In each processor clock cycle, both a memory access for data and a memory access for the instruction fetch are required. With only one set of busses to memory, the *Reconciler* must run at twice the speed of the processors.

When errors occur, the data that describes this error - the Error Syndrome - is stored in memory. The device that handles the collection of this data is the Error Syndrome Storage Device (ESSD). It also interfaces with the *Reconciler*, as all memory reads and writes pass through it.

A state machine, called *Interrup*, was built to coordinate error interrupts and the clock controls. *Interrup* generates a TRAP external to the processors and forces them into the Interrupt Service Routine (ISR). While the Error Syndrome is being stored in memory, the clock to the processors is stopped. With the processors not running, the only bus demand will be for the Error Syndrome.

This research fully realized the processor/voter interface. It also built the Reconciler, which controls memory interface. The research also created the ESSD to record and control storage of Error Syndromes. Finally, the Finite State Machine Interrup was built to control the various states the CFTP can be in.

Further research is required to determine memory requirements and other off chip requirements for the full CFTP design. The interrupt service routine also needs to be written.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The space environment is extremely hazardous to electronic systems. The lack of atmosphere, extreme temperature variations and radiation effects are all issues that electronic systems designers must contend with when building their systems.

Another concern that designers face is the fact that once their system is deployed, correcting any mistakes or upgrading the system it is very difficult, if not impossible. The infamous satellite that was sent to Mars and crashed due to an error in calculating in English instead of metric units is a perfect example of a mistake that could not be corrected once the satellite was launched. Additionally, being able to upgrade the system hardware on satellites requires a great deal of effort. The satellite must be captured by either the Space Shuttle or another satellite, and then the old systems removed and new ones installed. This is obviously both very expensive and extremely complex.

The focus of this thesis is on the construction of the Configurable Fault Tolerant Processor (CFTP), a design that can withstand the radiation hazards of space and allow for processor upgrades in orbit, without requiring a rendezvous or other complex evolution to perform the upgrade. The CFTP uses a Triple Modular Redundant (TMR) fault tolerant scheme in order to mitigate Single Event Upsets (SEUs), and is instantiated on a Field Programmable Gate Array (FPGA), which is a reconfigurable logic device.

The effects of radiation are numerous, and well documented in other writings. For the purposes of this research, the chosen hardware is assumed to be hard enough for the total dose and dose rate effects of the orbits of interest, so the focus is on Single Event upsets and their mitigation.

A. SINGLE EVENT UPSET HISTORY

Table 1 shows the various effects and the techniques used to protect against them. The radiation effects of most concern to space system designers are total dose, Single Event Latchup (SEL) and Single Event Upset.

Total dose effects are the cumulative effects of radiation on an electronic device over its lifetime. As charged particles impact the device, they degrade its performance a tiny amount. As the particles' effects accumulate, the degradation will eventually render the chip non-functional. There is no way to overcome total dose effects, only delay the end of life of the chip. This is done through various techniques, some of which are shown in Table 1.

Radiation Effect	Mitigation Techniques
Total Dose	Radiation-Hardening Silicon-On-Sapphire Silicon-On-Insulator Thin-Gate-Oxide Shielding
Single Event Latchup (SEL)	Radiation Hardening Guard Rings
Single Event Upset (SEU)	Quadded Logic Software Fault Tolerance Triple Modular Redundancy (TMR)

Table 1. Radiation Effects and Mitigation (From Ref. [1].)

Single Event Latchup is when a charged particle forces a transistor on the chip to remain in one state. If the transistor is left in the latchup state long enough, or with a high enough charge, it can burn out. The mitigation techniques for preventing SELs are shown in Table 1.

The FPGA chosen for implementation of the CFTP design is total dose tolerant to 100 kilorads, which is sufficient for the one year minimum flight duration. The FPGA is also SEL immune.

A Single Event Upset is an event which is initiated by a charged particle passing through a transistor. If the charged particle causes a state change to be latched into a register or flip flop, a bit flip occurs [1].

These bit flips can cause serious harm if not corrected. Imagine the thruster of a satellite being activated by a bit flip. Various techniques to counter SEUs have been developed over the years. One such method is a TMR scheme.

Previous NPS research led to the concept this thesis is implementing. The original TMR design had three separate hard-core processors whose output was sent through a hardwired voter [3]. As research progressed, a determination was made that using an FPGA for the hardware implementation would allow for all three processors to reside on the same chip as the voting logic [1]. This decreased both the size and power constraints of the overall system. It also allows for reconfiguration after hardware build, which is a distinct benefit.

B. MICROPROCESSOR

This version of the CFTP uses a KDLX 16-bit Reduced Instruction Set Computer (RISC) as the microprocessor [2]. The processor is a soft-core version of the hardware design [4]. A soft-core design is a firmware program that is intended for implementation on a programmable logic device. The primary advantage of this type of design is that is can be either corrected or upgraded simply be rewriting the design code. In contrast, hard-core processors are hardwired transistors and wires which cannot be changed after construction.

C. VOTING LOGIC

Triple Modular Redundancy is a majority voting scheme. As seen in Figure 1, three devices send their output into the voting logic. The majority of the device output is the voted output.

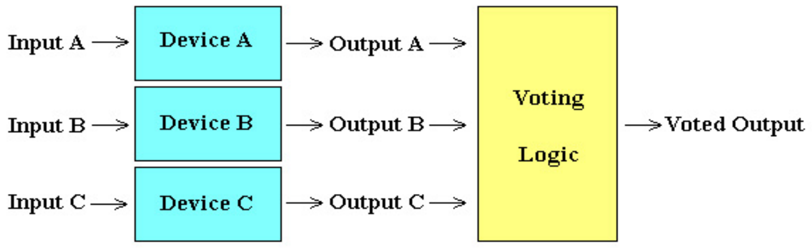


Figure 1. Basic TMR Concept (From Ref. [1].)

Figure 2 shows how the TMR concept is applied to a microprocessor. This research is designed as a System on a Chip (SOC) where all of Figure 2, with the exception of memory, is instantiated on a single chip.

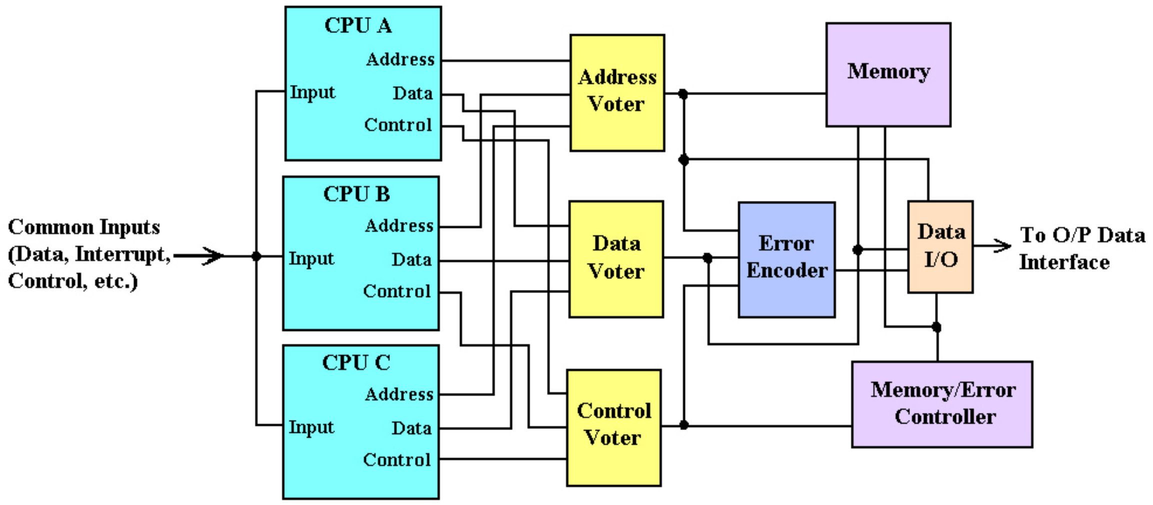


Figure 2. Microprocessor TMR Concept (From Ref. [1].)

In Figure 2, the three devices from Figure 1 are microprocessors. These microprocessors all run the same program at the same time. The outputs of the processors are the inputs of the voters. In the case of the CFTP, there are discrete voters for each of the processor outputs.

The voting system used to implement the TMR portion of the design is expanded from previous NPS research [1]. The single-bit voter designed by that research has been replicated to account for the multi-bit busses used by the KDLX processor.

D. MEMORY CONTROLLER

One of the goals of CFTP is to allow for upgrading the processor over time. In order to allow for upgrades, a design constraint was given for the on-chip design. Interface to the off-chip memory is to be by a single address and data bus pair. The KDLX processor has a program counter bus, instruction bus, address bus, and data bus; in order to consolidate the inputs and outputs, a memory controller was added to allow for the instruction and data to share one bus off the chip, and the program counter and address to share the other.

E. PURPOSE

The main purpose of this research is to implement a TMR design on a radiation tolerant FPGA. The successful implementation would indicate a potential solution to launching advanced microprocessors into space. It would also allow for the upgrading of these processors while on orbit.

F. ORGANIZATION

Chapter II is a description of the realm of FPGAs, soft-core processors and the one chosen for this research. Chapter III is a description of the voter, its development and implementation. Chapter IV explains how the clock is controlled and errors are handled. Chapter V discusses the integration of the individual components. Chapter VI contains conclusions and follow-on research recommendations.

G. ADDITIONAL DOCUMENTATION

Appendix A contains full design schematics and the hardware description language for the implementation of the CFTP design. These schematics and the code are not directly referenced throughout the thesis in order to present a smooth flowing format.

The CFTP design is currently scheduled to be launched on two satellites. One is the Naval Postgraduate School Satellite (NPSAT); the other is a U. S. Naval Academy sponsored satellite, MIDSTAR. In order to have the CFTP onboard these satellites, the experiment had to be reviewed by the Space Experiment Review Board (SERB). The preparation for this review process occurred concurrently with the development of the design. Appendix B describes the Department of Defense (DOD) SERB process. The research on this thesis was conducted while submitting the CFTP to the SERB process. This greatly facilitated the understanding of the DoD Space Program as a whole, and the role of NPS in that program.

THIS PAGE INTENTIONALLY LEFT BLANK

II. FIELD PROGRAMMABLE GATE ARRAYS

The flexibility that a Field Programmable Gate Array (FPGA) provides to a system designer is tremendous. The device can be reprogrammed a virtually unlimited number of times, and reprogramming can be done simply by the end user, hence the "field programmable" portion of the title. In addition, current generations of FPGAs contain large numbers of gates. The Xilinx Virtex XCV200 contains 236,666 systems gates [5], and the Xilinx Virtex II XC2V8000 contains 8 million gates [6]. These large gate counts allow for complex architectures to be implemented on the devices. Many FPGAs are large enough that complex microprocessors can be instantiated on the chip. These emulated processors are called soft-core processors. This Chapter will detail how FPGAs are used in emulating microprocessors, how the CFTP uses this technology to implement the design for spaceflight, and the reasons behind the choice of the processor for this research.

A. FIELD PROGRAMMABLE GATE ARRAY COMPOSITION

FPGAs evolved from simple programmable logic arrays (PLAs). A PLA is a series of AND-OR gate combinations which can be configured to emulate logic circuits. The disadvantage to a PLA was that once it was programmed, it was fixed in that configuration.

As the desire to emulate more complex circuits increased, more complex devices were created: the Programmable Logic Device (PLD), and then the Complex Programmable Logic Device (CPLD). In a PLD, the circuits are denser, and they can be reprogrammed. But the devices typically must be removed from the overall design for reprogramming. A CPLD is simply a collection of PLDs, with programmable interconnect between each PLD.

After CPLDs came FPGAs. In an FPGA the structure is very different from a CPLD. The FPGA is not a collection of PLDs, but a collection of smaller logic blocks. Each logic block is interconnected to the other logic blocks on the chip. A portion of the overall design can be implemented in the individual logic blocks, with the interconnect system passing signals between the blocks.

Figure 3 shows the differences between a CPLD and an FPGA. Part (a) shows a Complex Programmable Logic Device (CPLD). Part (b) shows an FPGA. While the FPGA has smaller logic blocks than the CPLD, it has a much larger number. The interconnect between these logic blocks allows for the ability to instantiate larger circuits.

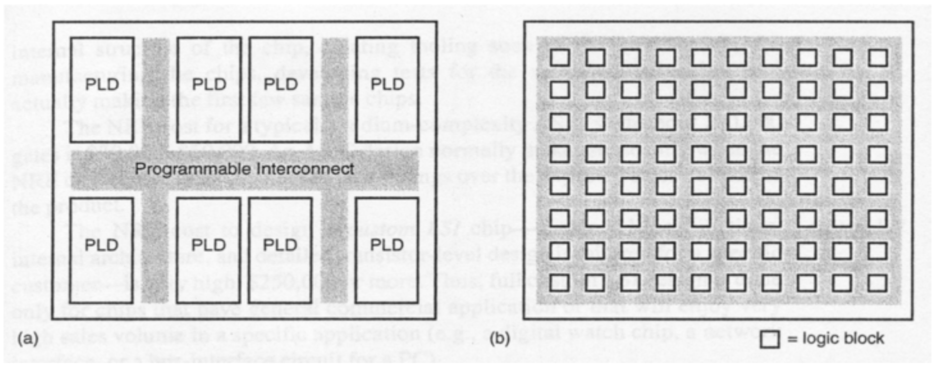


Figure 3. PLD and Interconnect Schemes (From Ref. [1].)

While a full description of an FPGA is not the focus of this thesis, a short background description of the FPGA architecture is necessary. For more information on FPGA architecture, readers are directed to the RG0References listed in this Chapter.

In an FPGA there are several different subsystems. The ones of direct interest for this thesis are the Configuration Logic Block (CLB), Input/Output Block (IOB) and the programmable interconnect [6].

The CLBs are where the design resides. Whether the design is something as simple as a 2-input logic device, or as complex as a microprocessor, the CLBs are configured to emulate the design. Each CLB is capable of being connected to other CLBs through the programmable interconnect systems so, if the design in question does not fit into a single CLB, the design can be parsed to multiple CLBs for full implementation and is transparent to the user.

Input/Output Blocks are used to interface with off-chip systems. One of the prominent considerations for the CFTP design was the limit of IOBs. While being able to design a system that can fit inside the complement of CLBs, without sufficient access to data off the FPGA, a design is worthless.

The FPGA chosen for CFTP was a Xilinx XCV800, which contains the equivalent of 888,439 gates [6]. One of the reasons this FPGA was chosen was for its external interface design. Xilinx FPGAs are available with two types of pin connections, flat-pack and ball grid array (BGA). A flat-pack resembles a traditional chip design, with the individual pins on the edge of the device, and is the type of interface of the XCV800. This type of pin connection has been used is space for years and is highly reliable. Unfortunately, newer, larger capacity FPGAs use BGAs.

A ball grid device has balls of solder in a grid pattern on the bottom of the chip. Inside these balls are the interface connections for the chip. The chip is soldered onto the printed circuit board by way of these solder balls. The process for attaching a BGA onto a board is complex and would require NPS to contract out for the manufacture of a system. Many of the connections are located on the interior of the array of balls and determining if they were all properly affixed is not feasible for space applications [7]. Finally, BGA technology is not fully spaceflight certified, based on concerns about the effects of space on ball grids. A detailed description of the decision making process for this choice of FPGA can be found in Reference [1].

B. SOFT-CORE PROCESSORS

When a microprocessor is instantiated on an FPGA, it is called a soft-core processor. This is as opposed to a hardwired microprocessor such as those found in desktop computers. The soft-cores are simply hardware description language (HDL) code which programs the various parts of the FPGA to act as the real processor would.

There are a variety of soft-core processors available. Some are free from companies, and others can be quite expensive. The various designs available go from simple RISC to x86 style processors. But most of them are intellectual property or proprietary software.

Essentially, any of these proprietary processors would be a black box, with all internal parts unreachable. One of the design issues for CFTP is the ability to manipulate the entire FPGA and design. For example, if a portion of the FPGA is damaged by radiation, full access to all portions of the design allows the possibility to rewrite the code to avoid that portion of the chip. Another reason for being able to access the internal portions of the soft-core is the ability to then place voters at the different stages of the pipeline. The decision for now is to have voters only at the output of the processors, based on radiation levels expected in the orbit of concern. But if these predictions are incorrect, voters may be required internal to the processors. This would not be possible with proprietary soft-core processors.

C. KDLX PROCESSOR

The KDLX processor is an HDL coded version of the DLX processor originally described in Hennessy and Patterson's *Computer Architecture: A Quantitative Approach* [4] and coded by Dr. Kenneth Clark [2]. It is a pipelined 16-bit Reduced Instruction Set Computer (RISC). The focus of this thesis is not on the details of the RISC architecture, nor on the design of the DLX computer. For information on these topics, readers should look at the Hennessy and Patterson book referenced above.

There were several reasons this processor was chosen, the largest being that the processor had already been designed and tested. This processor design was originally implemented for an NPS dissertation [2]. This dissertation was on the development of a model to predict the SEU tolerance of complex digital systems. The digital system used in the testing was the KDLX processor.

Another reason was the problems associated with the issues of proprietary software enumerated previously. A third factor for choosing this processor was its size. It is small enough that three processors easily fit onto the FPGA, along with all the other portions of the design required for the full CFTP.

Being a simple RISC processor, there is ample software and information regarding this processor available that is not proprietary. The architecture of this type of processor is also taught at NPS, and therefore it is a good processor for an experiment that will be worked on by several students over the course of the experiment lifetime.

The design of the DLX processor presented an interesting dilemma. The DLX processor is a Harvard architecture, which means it has separate address and data busses for instructions and data. That design is intended to be implemented with separate memories for data and programs.

However, one of the initial design decisions made for the CFTP was that there would only be one data bus and one address bus between off-chip memory and the FPGA. With twice the busses required for the KDLX processor, some innovative thinking was required to surmount this problem. The solution is described in detail in Chapter IV.

D. CHAPTER SUMMARY

This Chapter explained the basics behind how FPGAs operate, and how a soft-core processor is instantiated. It also described the particular processor chosen for this thesis, and the reasons for that decision. The next Chapter will describe the evolution of the voter and its incorporation into the full CFTP design.

THIS PAGE INTENTIONALLY LEFT BLANK

III. TRIPLE MODULAR REDUNDANT ARCHITECTURE

The intent of Triple Modular Redundant architecture is to allow the outputs of three devices to be compared in a majority voter. The reason it is called modular is because the inputs to the voter are independent of each other. In the case of the CFTP, these inputs come from the three separate processors, which do not directly interact with each other. This Chapter will explore the development of the voter used in the CFTP.

A. OVERVIEW

The KDLX processor, shown in Figure 4, has six outputs. There are three single-bit outputs, for program counter read, data read, and data write. There are two 16-bit outputs, the program counter and data address. The data bus is a bi-directional bus. The three copies of the KDLX processor are connected to the inputs of the voters, the busses on a bit-by-bit basis. If any of the 51 bits do not match between the processors, then an interrupt will be generated. The interrupt service will be described in Chapter IV.

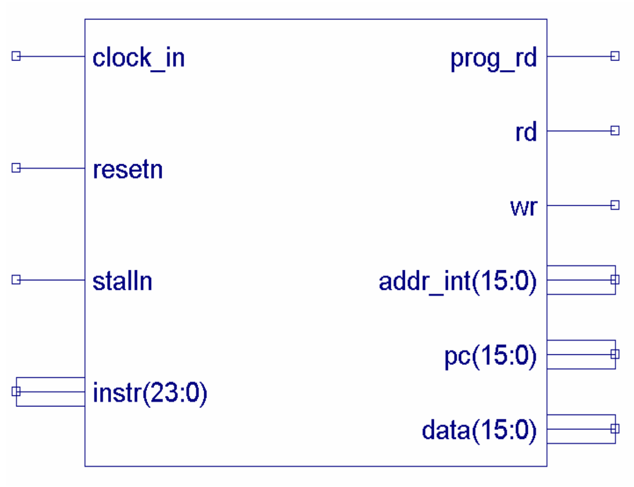


Figure 4. KDLX Processor

Figure 5, the TMR Assembly (TMRA), shows the three processors on the left, and the voters on the right. Each voter has four outputs: Voted Result, Error, CID_0 and CID_1. CID_0 and CID_1 are a two bit status bus that identifies the processor in error. More detail on the CID bits is given in Section B in this Chapter.

Once the data has been voted on, the voter outputs are consolidated into four busses. All of the same type of output is on a single bus, the error signals on one bus, the voted data on another, etc. The Voted Result Bus is sent to memory, while the remaining three busses are then used by the error handler.

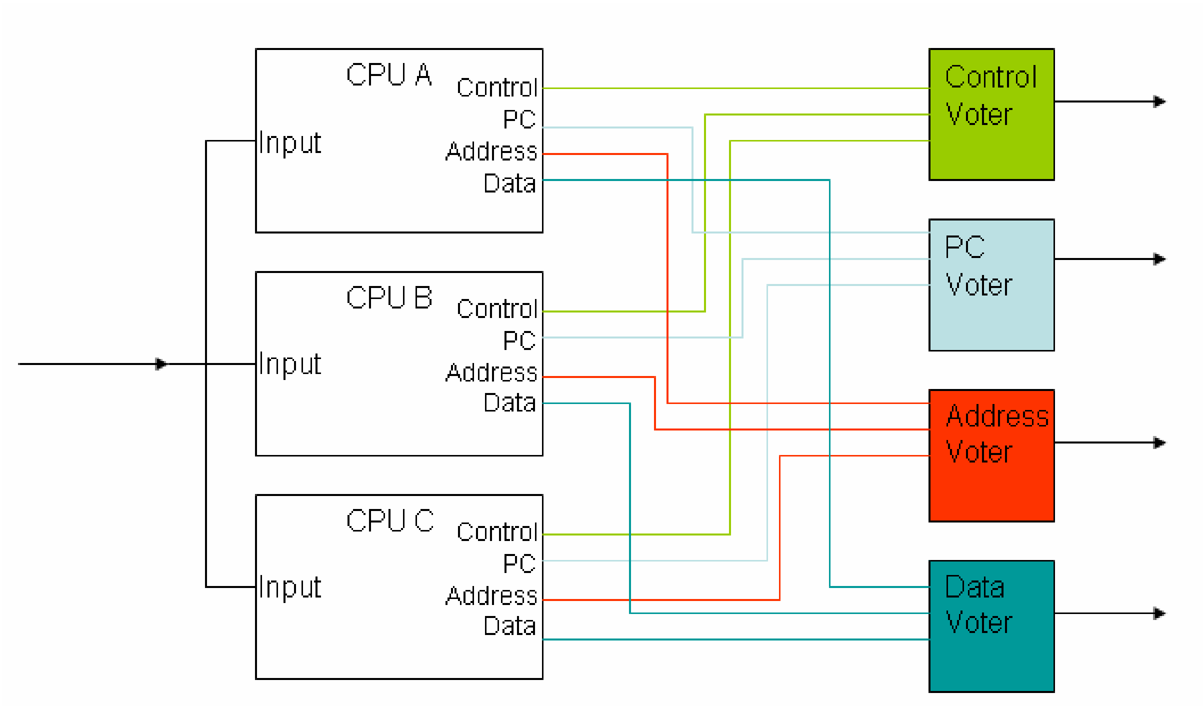


Figure 5. TMR Assembly (TMRA)

B. VOTER DEVELOPMENT

A basic single bit voter takes three inputs, A, B, and C, and enters them into the logic circuit shown in Figure 6. The output, Y, is then the "voted output." Each input is ANDed with each of the other two inputs. This circuit yields a majority result; if two of three inputs are high, the output will be high. In the CFTP design, the voter inputs are individual bits of the outputs of the processors.

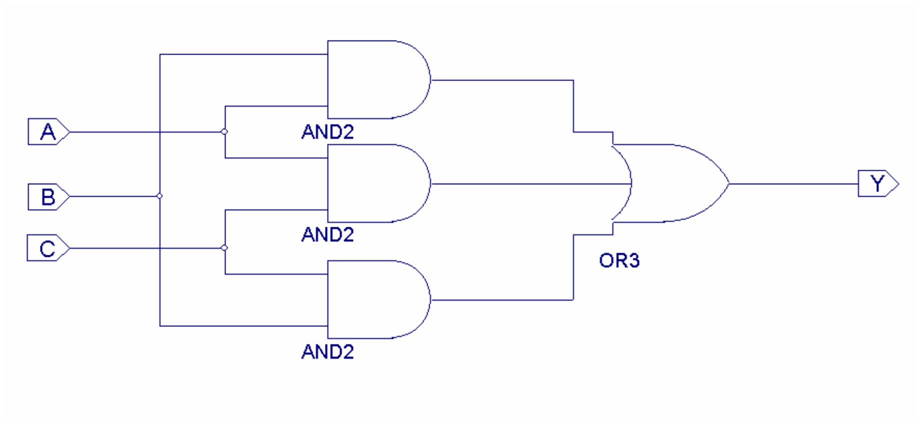


Figure 6. Basic 1-Bit Voter Circuit

The next item required is an indication that an error has occurred, so correction can be accomplished. Figure 7 shows a single bit voter with data error detection logic built in. The 3-input AND gate and the 3-input NOR gate will produce a high output when the three inputs are the same. When any of these outputs are different, this indicates one of the three inputs is different, and the resulting output "ERR" will asserted.

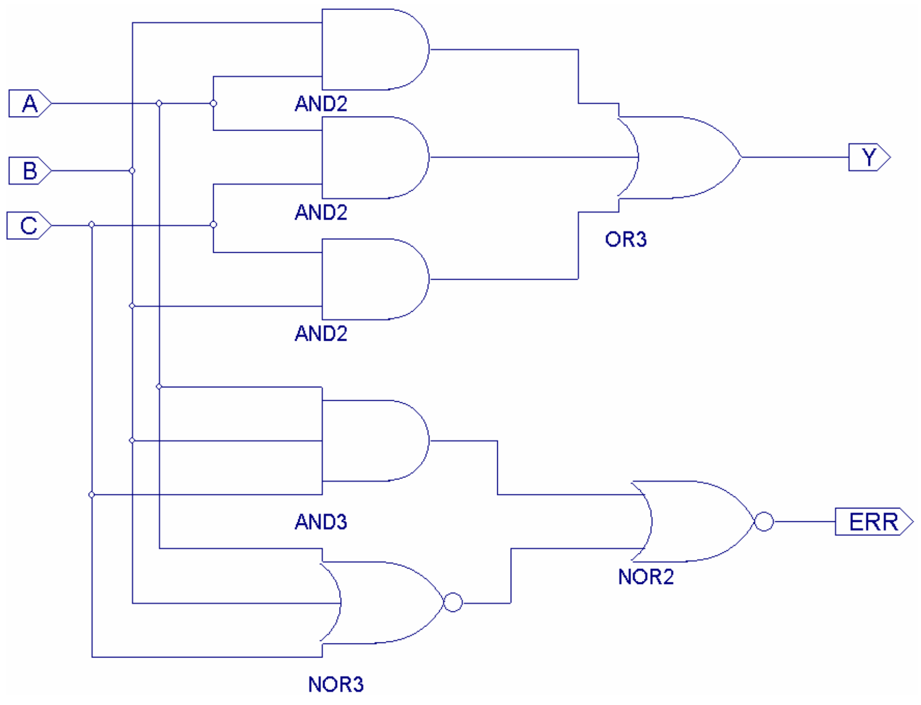


Figure 7. 1-Bit Voter with Data Error Detection (After Ref. [1].)

If we desire to know whether an error is actually in a processor, or is happening in the voter circuit itself, we need to replicate the voter. Figure 8 shows two voters with a voter error indication. In the event that an SEU occurs in the voter, it may be helpful to realize the error is not a data error but rather a voter error. The voter error is detected by duplicating the voter and using an XOR gate to compare the two voted outputs.

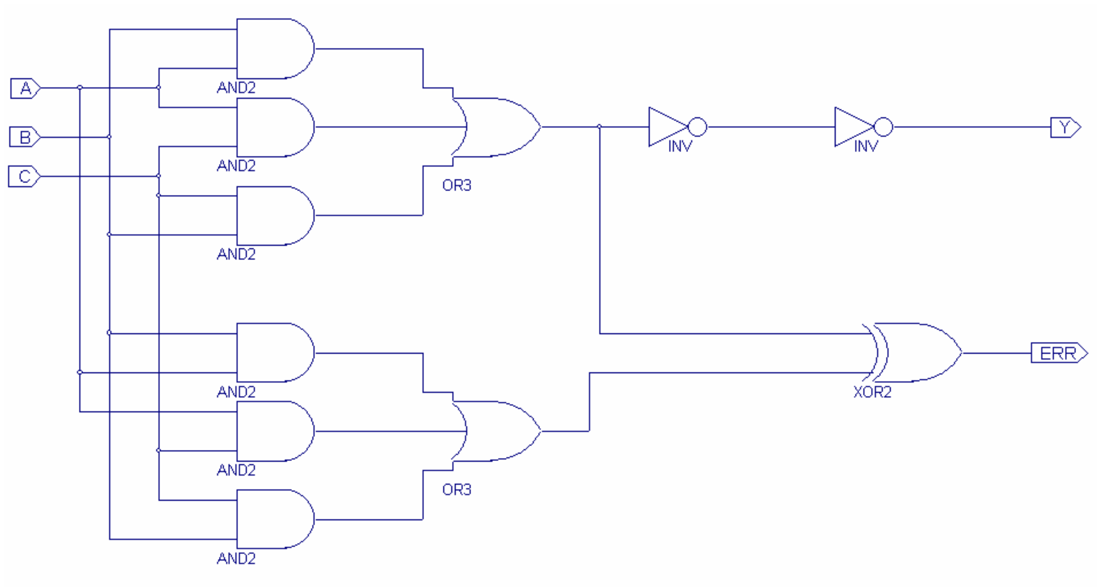


Figure 8. 1-Bit Voter with Voter Error Detection (After Ref. [1].)

If the error occurred in the voter circuit, no corrective action may be required, beyond the act of voting the data and recording the error location. If the error came from a processor output, the processor fault which caused the error must be corrected.

In a data error, determining which processor is in error is required. Figure 9 combines the data error and voter error configurations and includes two more outputs, which are used to determine which processor was in error in the event it was a data error. This full voter was the single-bit design proposed for use in the CFTP in earlier research [1].

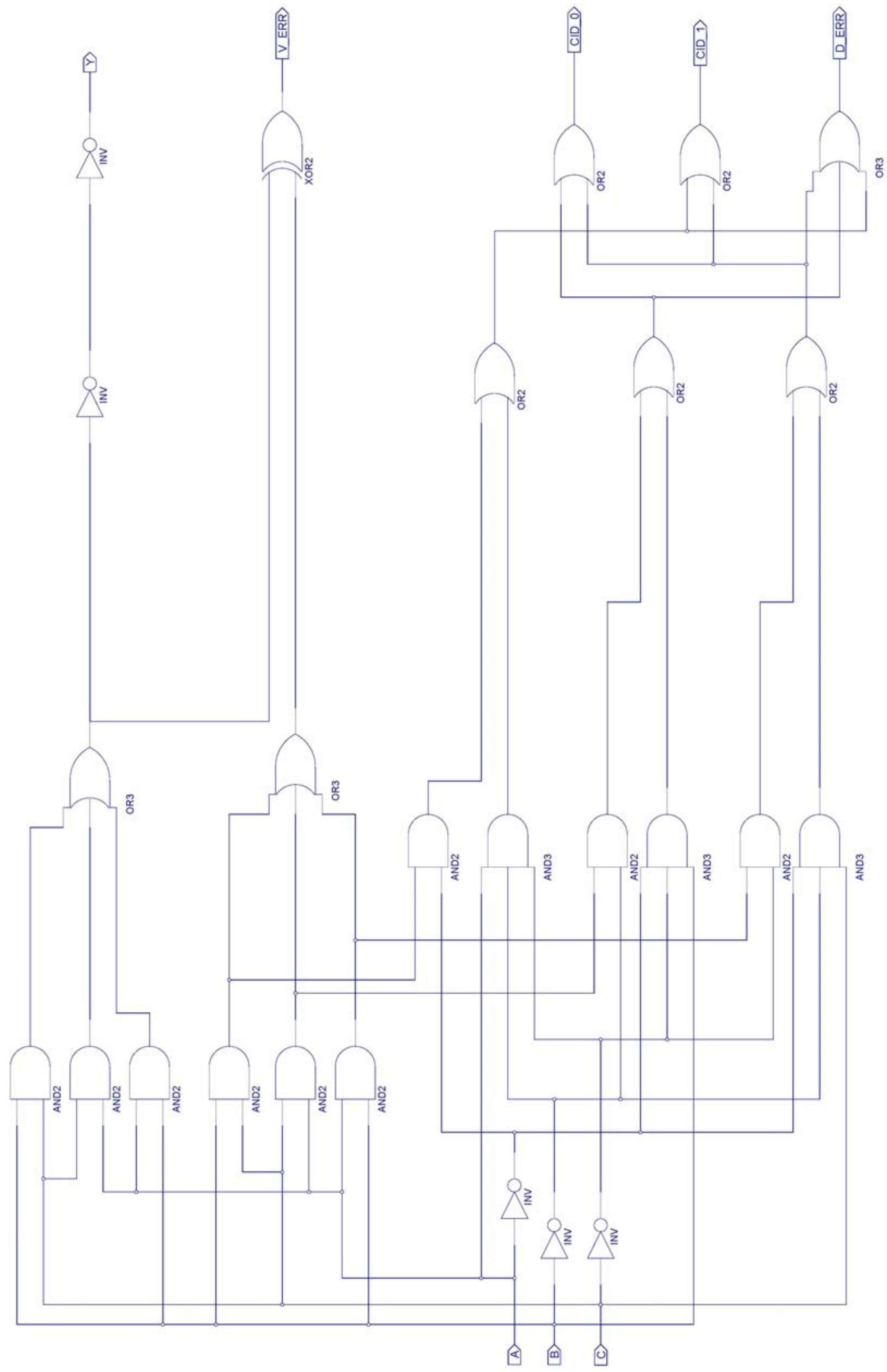


Figure 9. 1-Bit Voter Circuit with Data- and Voter-Error Detection and Location (After Ref. [1].)

After some deliberation, it was decided that the need for voter versus data error discrimination was not necessary. It was initially believed that determining a voter error would save time in the interrupt service routine. Since the data was assumed to be correct, the interrupt would only require storing the error syndrome. But if the voter is in error, the data still may be corrupted in the process. A second reason for eliminating error discrimination was error collection. Having a second error signal also requires additional requirements for collecting the data on these errors. Instead of having to store 51 bits of Error, it would be 51 bits of Data Error and another 51 bits of Voter Error. Finally, discriminating a voter error increases the complexity and size of both the voter and the logic required to process the error.

With all these considerations in mind, Fig. 9 was modified to create Fig. 10, which has a single voter, the error indication, and error location. Table 2 shows how the voter indicates which processor is in error.

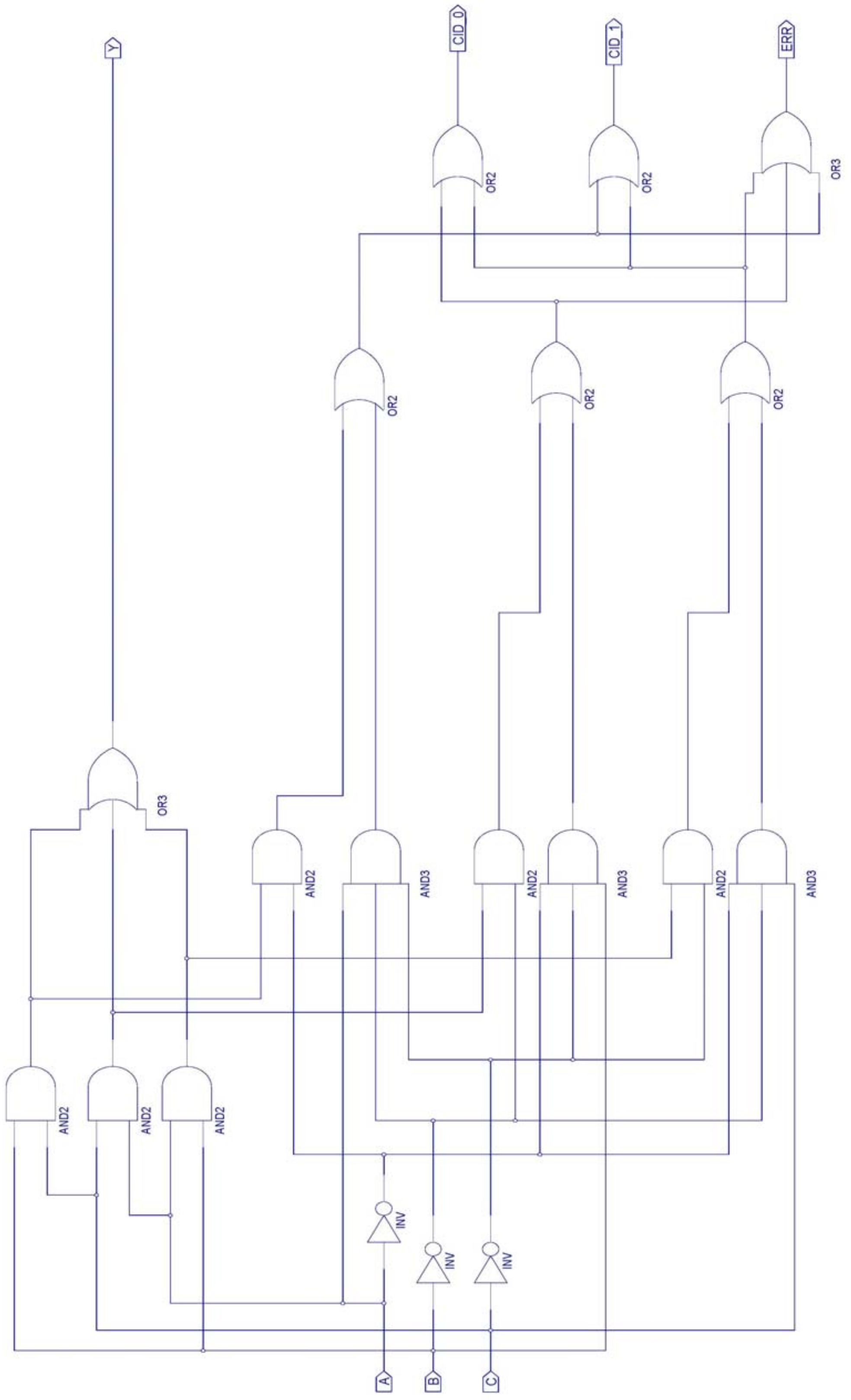


Figure 10. 1-Bit Voter with Error Detection and Location

CID_0	CID_1	Error Location
0	0	None
0	1	Processor 1
1	0	Processor 2
1	1	Processor 3

Table 2. Error Location Table

C. COMBINATION OF VOTERS AND PROCESSOR

For the processor chosen for the CFTP, a 16-bit RISC processor, this voter had to be replicated 16 times for each of the 16-bit busses. The voter inputs come from the processor busses and outputs are placed onto a 16-bit address or data bus.

The CFTP processor has three 16-bit busses that require voting, so in essence 48 voters needed to be created. In addition to the bus voters, there are three single bit outputs that require voting: the strobes for the program counter, data read and data write. So a total of 51 voters are required.

D. CHAPTER SUMMARY

This Chapter presented the evolution of the voter from a simple single-bit voter to the robust voter with error identification and reporting. It also described the configuration of both single-bit and 16-bit voters with respect to the three processors. The next Chapter will explain the clock control system and how errors are handled.

IV. CLOCK CONTROL, ERROR HANDLING, AND MEMORY INTERFACE

This Chapter will discuss the clock control method, error handling system and the interfacing with off-chip systems for the CFTP. The design constraints of the CFTP required unique solutions. The clock used for the processors had to be different from the remaining systems in the CFTP. At the same time, the error handling routine had to be capable of operating in an environment where some of the components operated at a different speed than others. Finally, interfacing with off-chip systems required a device that could coordinate the various bus demands and speeds.

A. OVERVIEW

The KDLX is a Harvard architecture design. This means it has two separate memories, one for instructions and one for data. The interface between the processor and these memories is four separate busses: a Program Counter bus (processor output), an Instruction bus (processor input), a Data Address bus (processor output) and a Data bus (bi-directional). The interface between the FPGA and the off-chip memory in the CFTP is a Von Neumann architecture. The Von Neumann architecture only has one address bus and one data bus, which are shared between the program and the data. Interface circuitry and a clock control state machine were needed to allow the KDLX to time share the memory interface. With this constraint to the design, a system was developed to coordinate the bus demands for the CFTP. It will be described in Section C of this Chapter.

The CFTP operates in several conditions. In a normal condition, the processors are executing instructions and the voters are checking the outputs. When an error occurs, the CFTP changes states from normal operation to the Error Syndrome Saving condition. Once the error is saved to memory the CFTP shifts to the Error Correction Interrupt Service condition. After the Error Correction is complete, the CFTP reverts to normal operation.

B. STATE MACHINE CONTROLLER

A state machine was built to control the states in which the CFTP operates. Figure 11 shows this machine, called *Interrup*. In normal operation, the processors require up to two transfers, each needing both busses. As stated earlier, there is only capacity for one address and one data transfer off-chip. This is solved by reducing the clock rate of the processors to half that of the rest of the design. Further detail on how this was accomplished is discussed in Section C of this Chapter.

With two clock cycles for bus transfer off-chip to every one of the processor clock cycles, the requirement to coordinate the bus demands is met. As seen in Figure 11, the states NormInst and NormData are the two normal condition states. During NormInst, *Interrup* asserts the output InstrAccess and deactivates the output DataAccess. During NormData, the outputs are reversed. This is because the KDLX processors will be asserting Program Read and Read/Write signals during both states, and only one of these transfers can occur at a time. These outputs are used by a device called the *Reconciler*, which controls memory accesses by all on-chip components.

In NormData, the state machine also looks for the input E, which is the error signal from the voters. When E is asserted, *Interrup* shifts from normal operation to the Error Syndrome Save condition.

In the Error Syndrome Save condition, there are two sub-conditions, ErrSynSave and ISRErrSynSave. Both are designed to perform the same tasks, with one exception. In the ErrSynSave condition, there is an additional state which generates an instruction that will begin the Interrupt Service Routine (ISR). Further explanation of this will be given later in this Section.

As the voters generate outputs, these outputs are consolidated and sent to a device named the Error Syndrome Storage Device (ESSD). The outputs are collectively called the Error Syndrome. The Error Syndrome is a total of 104 bits: the 51 voted bits, the 51 error bits, and a consolidated CID_0 bit and CID_1 bit.

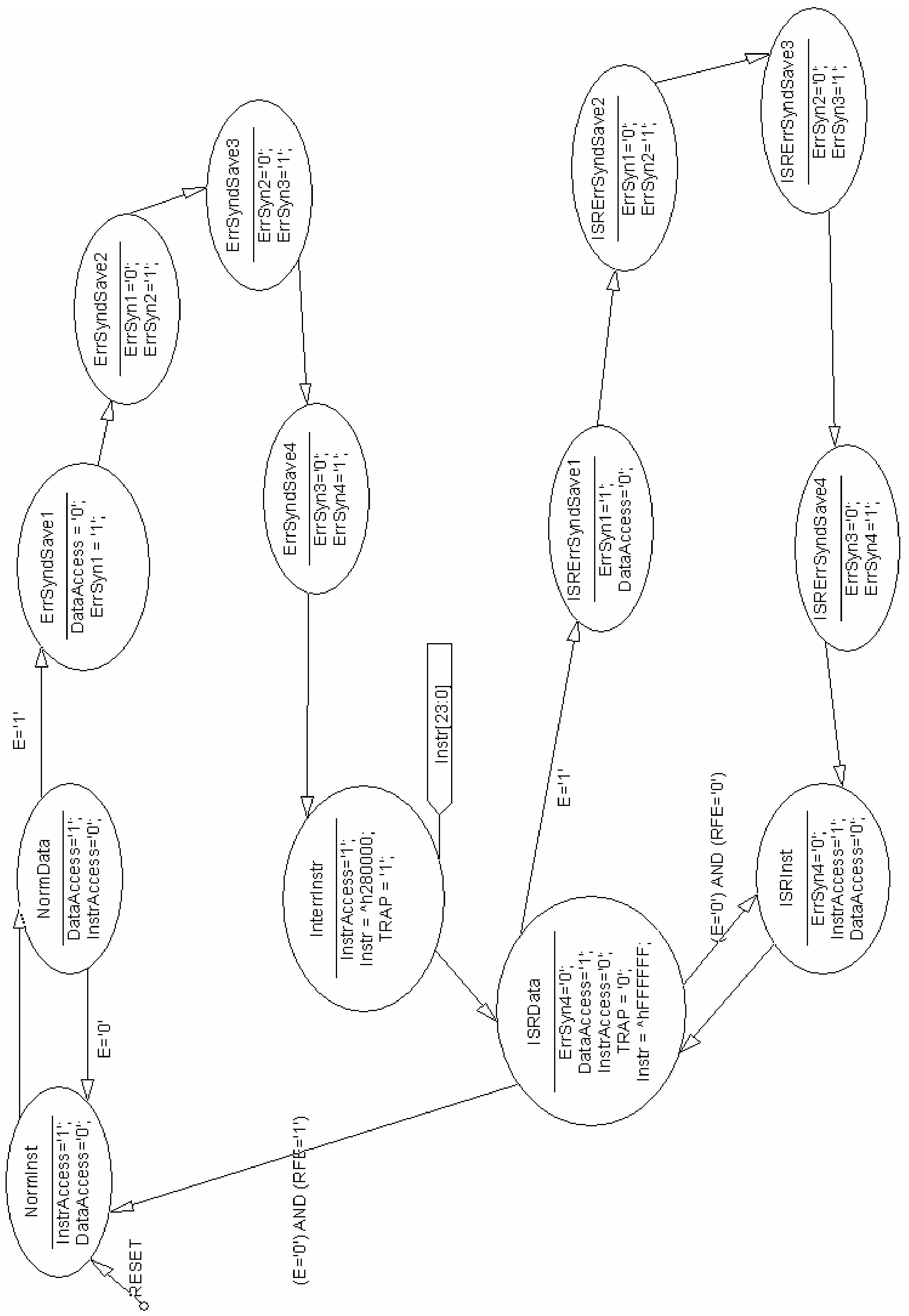


Figure 11. Finite State Machine Controller

The CID bits are each put through a 51-to-1 OR gate which is designed to reduce the amount of data that will be saved. With the assumption that only one error will occur at a time, the CID bits do not need to be saved in total. Further description of the ESSD operation will be given in a later Section.

With the 104 bits of data to be transferred to off-chip memory and a 32-bit data bus, four clock cycles are required. This is the reason for the four ErrSynSave states. Each of them asserts a corresponding ErrSyn signal which is used by the ESSD to port data to the *Reconciler*; the ErrSyn signals are also used by the *Reconciler* to stop the processor clock. The reason for this will be explained in Section C. Once the Error Syndrome Save is complete, the InterrInstr state generates the ISR instruction and asserts the signal TRAP. This signal is used by the *Reconciler* to port the TRAP instruction, seen in Figure 12, instead of the instruction coming from memory, to the processors. With the processor clock stopped, this instruction is not lost, simply put on hold until the ErrSyn signals are deactivated.

Instruction: TRAP (Software Trap)

23 16	15 0
Opcode:0x28	Immed

Figure 12. TRAP Instruction Description (From Ref. [2].)

The TRAP instruction will take the Program Counter contents and place it in the Interrupt Address Register, and then load the Program Counter Register with the immediate value. The immediate value is the address of the first instruction for the Interrupt Service Routine [2].

Once the TRAP instruction has been inserted, the CFTP moves to the Error Correction condition. In this condition, the Processors are restarted and the ISR begins. The ISR stores each register to memory and then reloads each register from memory. As the registers are stored, the data passes through the voters. If there are any errors, they are corrected. It is expected that at least one miscompare will occur during this routine, the error that caused the routine to occur in the first place. To ensure that errors which are detected once the ISR has begun do not start another ISR, a second set of Error Syndrome Save states exist.

Once the error correction is complete, the final instruction of the ISR will initiate a return to the normal states of operation. This is accomplished by the ISRData state looking for the input RFE, Return From Exception, which indicates the last instruction of the ISR has been loaded from memory into the processors. This signal combined with E not indicating an error will force the state machine back to normal operation.

C. CLOCK CONTROL AND MEMORY INTERFACE

The *Reconciler* generates at a clock rate which is half the rate of the input clock. This clock is an output of the *Reconciler* and an input to the TMR Assembly. But the need to stop the processor clock for an error required an additional control signal for the processor clock in the event of an error.

The reduction in clock speed is accomplished through a clock delay-locked loop, CLKDLL. As shown in Figure 13, the input to CLKDLL is the master clock. It outputs two clock signals, one at half the speed of the original signal and one at the same rate as the input signal. Both signals are kept in phase by a feedback loop [8].

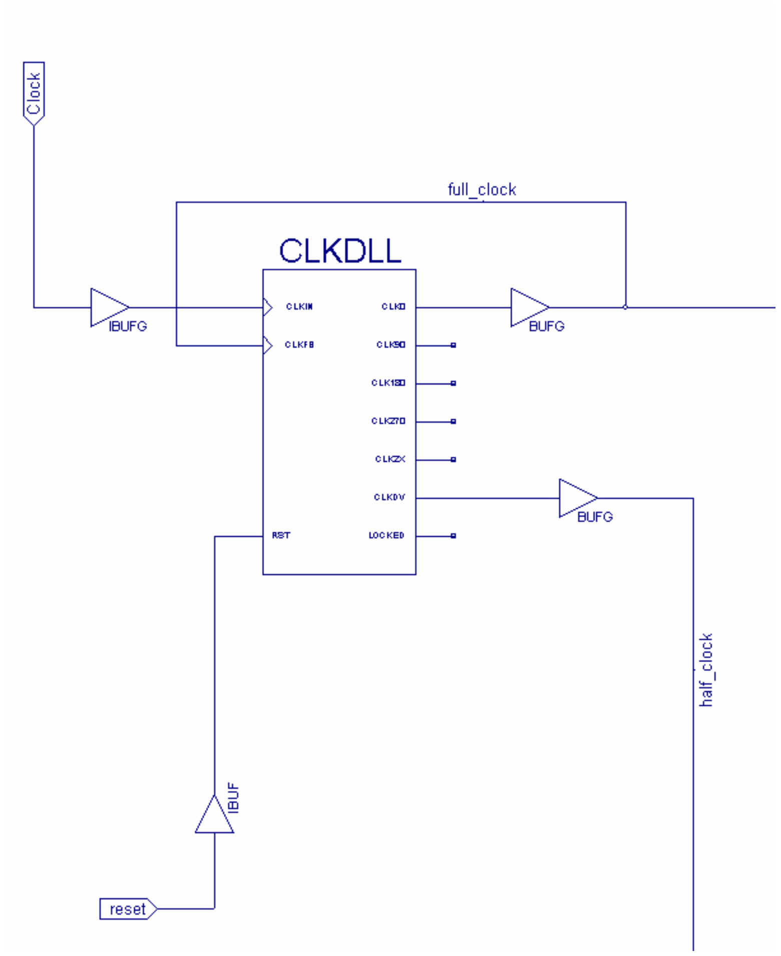


Figure 13. Clock Controller

The higher speed clock is used by the remaining devices in the CFTP, which includes the *Reconciler*, *Interrup*, and the ESSD. It is an output of the *Reconciler* called rest_clock.

In Figure 14, the processor clock signal is the input to a tri-state buffer, which takes its enable input from the signal ErrSyn. ErrSyn is the OR of the signals ErrSyn1 through ErrSyn4, from *Interrup*. If any of them are high, the tri-state buffer will be disabled, stopping the processor clock, and set the *Reconciler* for Error Syndrome transfer.

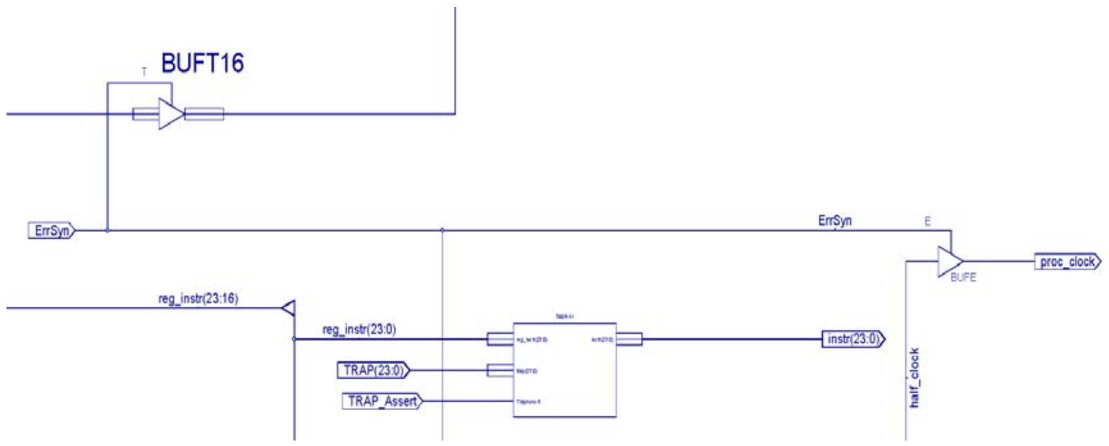


Figure 14.   Processor Clock Control

The *Reconciler* also controls the memory interface. It receives the inputs from the processors for reading and writing data, and reading the instruction, along with all the control signals the processors use. The ESSD also ports the Error Syndrome to the *Reconciler* for storage into memory. Using the clock control previously described, the *Reconciler* completes all transfers to and from memory.

D. ERROR SYNDROME STORAGE DEVICE (ESSD)

The ESSD is designed to capture the state of the processors when an error occurs. It stores the output of all the voters, and the error signals of all the voters. For each bit voted on, there is an error bit. There are 51 bits voted, so there are 51 error bits. In addition, there are two bits to determine in which processor the error occurred. In total, 104 bits are required to make up a single error syndrome.

All of this syndrome data is latched into the ESSD every clock cycle, regardless of an error occurring. It is only stored into memory when an error activates the Error Syndrome Save states of the state machine. The errors are saved to the highest address locations in memory, and are stored "backwards" by a down counter. Figure 15 shows the down counter and the ESSD outputs.

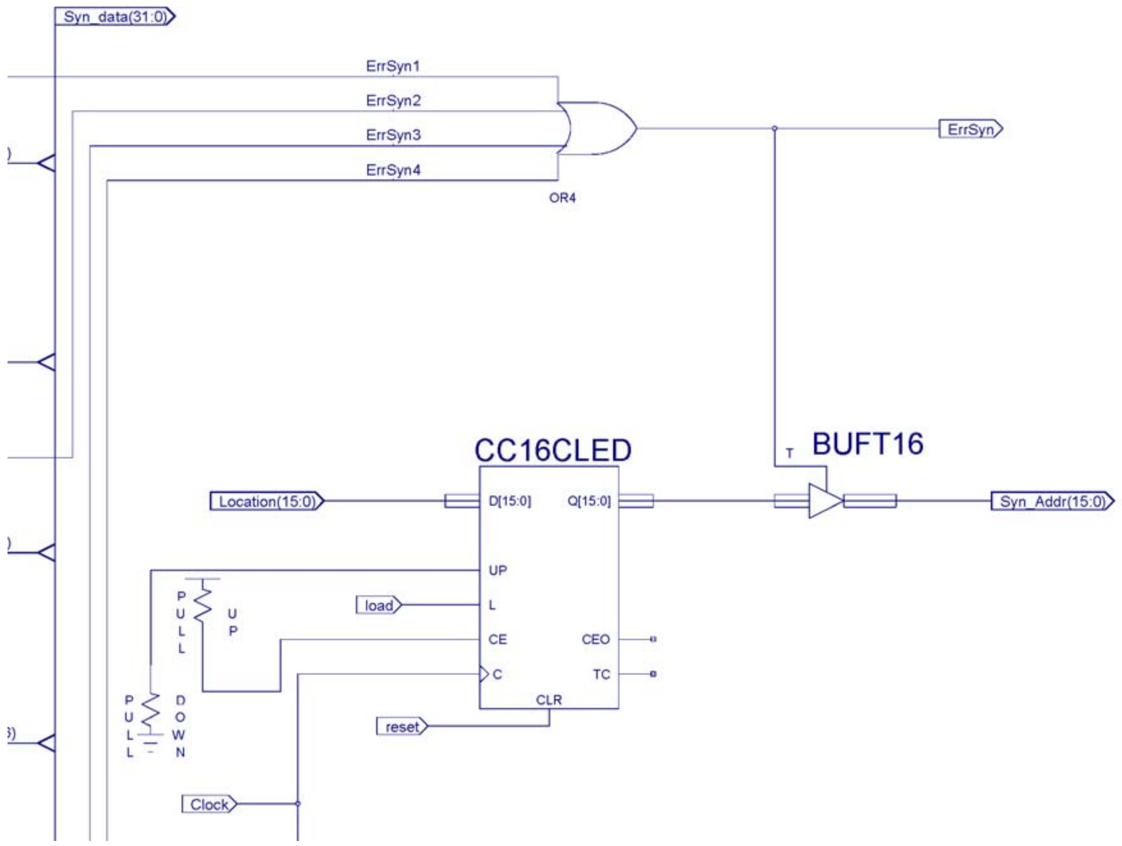


Figure 15. Error Syndrome Storage Device

The highest location in memory is loaded into the down counter via the input Location. When the first error syndrome is stored to memory, the first 32-bit word is stored in the last memory slot, the second word in the next to last, etc. As each word is stored, the address is decremented via the down counter and is output as Syn_addr. The location in memory that it begins counting down from can be loaded into the down counter by loading in the start address into the Location I/O marker and asserting the Load I/O Marker.

Figure 15 also shows the other two outputs of the ESSD. Syn_data is the 32-bit word to be stored, and ErrSyn is the combined signal of each of the four Error Save State signals. The four state signals are ORed together and both output, and used to assert the tri-state buffer with the syndrome address. These three outputs are input into the *Reconciler*.

E. CHAPTER SUMMARY

This Chapter described the error handling and clock control for the CFTP. The next Chapter will discuss the simulation results and the full integration of the subsystems of the CFTP.

THIS PAGE INTENTIONALLY LEFT BLANK

V. INTEGRATION

The integration of subsystems into a design, whether it is for a car, a house or an electronic system, can be difficult. This integration is further complicated when it is for a system that has never been built before. Such is the case in satellite design. Most satellites are unique in their mission, and therefore the subsystems onboard are unique, many having been built specifically for the mission.

The integration of the components for the CFTP was no different in this respect. The subsystems for the CFTP were built specifically for this mission, and they each needed to be tested separately and then integrated into a complete design. This Chapter describes how these subsystems were integrated.

A. SUBCOMPONENT TESTING

Each subcomponent needed to be tested with the synthesis tool used to build them. Testing was accomplished by checking for errors in the schematic design, followed by compiling of the design into Hardware Description Language. Each schematic is coded into HDL by the synthesis tool. As it is compiled, the compiler checks for errors in coding caused by improper design. For example, if the design uses a 32-bit bus, but only 16 bits are connected to a higher level design, the schematic check will not detect an error, but the compiler will.

As these pieces were finished, they were tested for proper circuit connectivity, and to ensure that the individual subcomponents would not exceed the FPGA capacity. Actual gate count capacity was not the largest concern. What was more constraining was the number of input/output (I/O) buffers used in the design.

As it turns out, the problem with buffer constraints was based on poor understanding of the Xilinx design tool. The actual buffer capacity was not a factor in design implementation. The synthesis tool in the Xilinx implementation software used I/O buffers for tri-state buffers at certain points in the design. This forced the design to require a larger number of I/O Blocks (IOBs) than the chip contained. The solution to the problem is explained in Section C of this Chapter.

Signals sent from one subcomponent to another required buffering for bi-directional busses or for sharing busses. In the case of the single data and address busses interfacing with off-chip memory, the two addresses from the KDLX processor had to be buffered opposite each other. For the TMRA, the data bus required several buffers in order to ensure that Read data did not interfere with Write data.

Redesign occurred each time the testing failed and an iterative process finally ensured that the individual subcomponents were functional.

B. INTEGRATION

Once these individual devices were built, they were combined into one full design, Figure 16. The inputs to the device Reset, Clock, Stall, and Load. Addr_out (16 bits) is the only output. Data_Instr (32 bits) is a bi-directional bus.

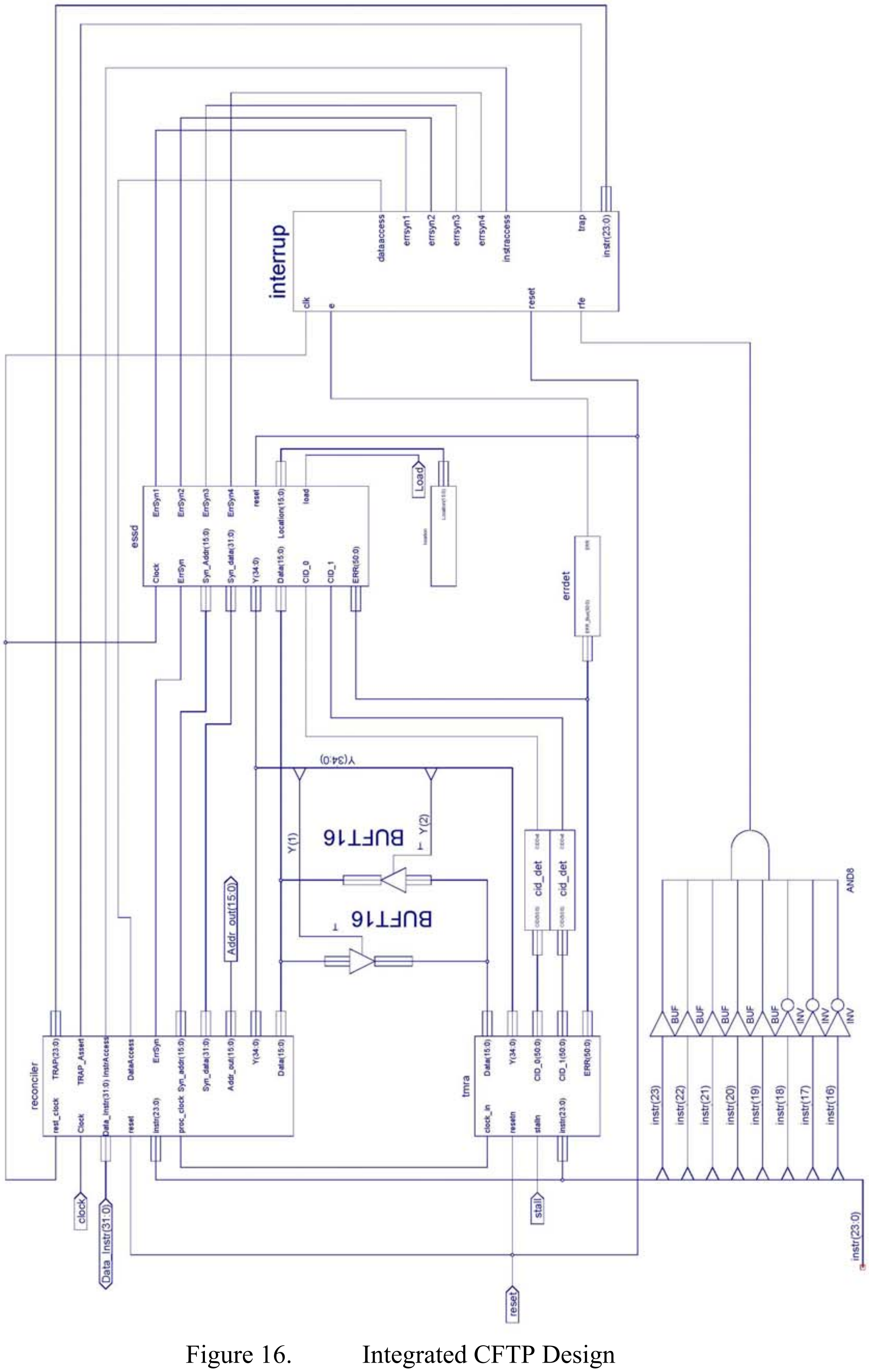


Figure 16. Integrated CFTP Design

The two tri-state buffers between the *Reconciler* and the TMRA are to control data to and from the processors. The control signals used for these buffers are the voted Read and Write signals from the TMRA.

The two macros shown below the tri-state buffers are 51-to-1 OR gates. They are intended to consolidate the Computer ID (CID) in order to reduce the number of bits to be saved in the error syndrome.

The same macro used for the CID busses was modified for use on the Error bus. The Error bus is consolidated since only one error signal is required to enter the ISR. The output is then used to start the Error Syndrome Save inside the *Interrup* macro.

At the bottom of Figure 16 is an 8-input AND gate. It is designed to detect the RFE instruction. When the opcode for the RFE is input into the processors, this signal will assert the RFE pin on the *Interrup* which shifts the CFTP from ISR operation back to normal.

C. FULL DESIGN TESTING

Once the individual components were connected together, more synthesis was required to ensure that signals from one device did not produce incorrect results in another. Problems occurred numerous times, but were typically easily solvable.

Most of the faults were caused by incorrect connection of the wiring. Others were caused by improper labeling of either inputs or outputs of subcomponents. But some faults were caused by serious design flaws.

One such example was in an iteration of the *Reconciler*. The initial design did not include a mechanism for injecting the TRAP instruction into the instruction stream. Once it was determined that the logical location to inject the TRAP instruction was inside the *Reconciler*, the design was reconfigured to allow for either an instruction to enter the stream from memory or to be injected from the FSM.

The redesigned *Reconciler* had two 24-bit tri-state buffers which would be enabled via the TRAP_Assert signal. Synthesizing the *Reconciler* by itself yielded no faults. But when the entire design was synthesized, a fault occurred. The design now required more buffers than the chip had capacity for. A subsequent redesign was done, whereby the two 24-bit buffers were replaced by a 24-bit 2-to-1 multiplexer. This brought the entire design back under the capacity limits of the FPGA.

D. INTEGRATION STATISTICS

Once the full design was synthesized, the design software provided a synthesis report, which detailed the uses of chip resources. A portion of that synthesis report is provided in Table 3.

Selected Device: v800hq240-6				
Number of Slices:	1678	out of	9408	17%
Number of Slice Flip Flops:	1699	out of	18816	9%
Number of 4 input LUTs:	2037	out of	18816	10%
Number of bonded IOBs:	36	out of	170	21%
Number of TBUFs:	425	out of	9408	4%
Number of GCLKs:	2	out of	4	50%

Table 3. Chip Resource Allocation

The number of slices is a representation of the gate count. In essence, only 17% of the entire chip gate count is used to implement the entire design. The other critical information from Table 3 is the number of bonded IOBs. An IOB is an Input/Output Block. This is where the pins on the outside of the chip interface with the design implemented on the FPGA. In the case of the CFTP, only 21% of the IOBs were used.

Both of these numbers indicate that there is ample room on the chip for a larger and more complex design. Since one of the long-range goals of the CFTP is to be reconfigurable, this is a good indication that this size FPGA will be capable of new configurations once it is in orbit.

E. PIPELINE CONCERNS

There are four stages to the KDLX pipeline [2]. When the KDLX processors are operating in normal mode, during each clock cycle an instruction resides in each stage. Figure 17 shows the pipeline timing when there are no errors. As discussed in Chapter IV, the memory is clocked at a rate twice that of the processor. So for every processor clock cycle, *P*, two memory clock cycles, *M*, transpire. At clock cycle *P*, instruction *X* enters the pipeline. At clock cycle *P*+4, instruction *X* exits the pipeline.

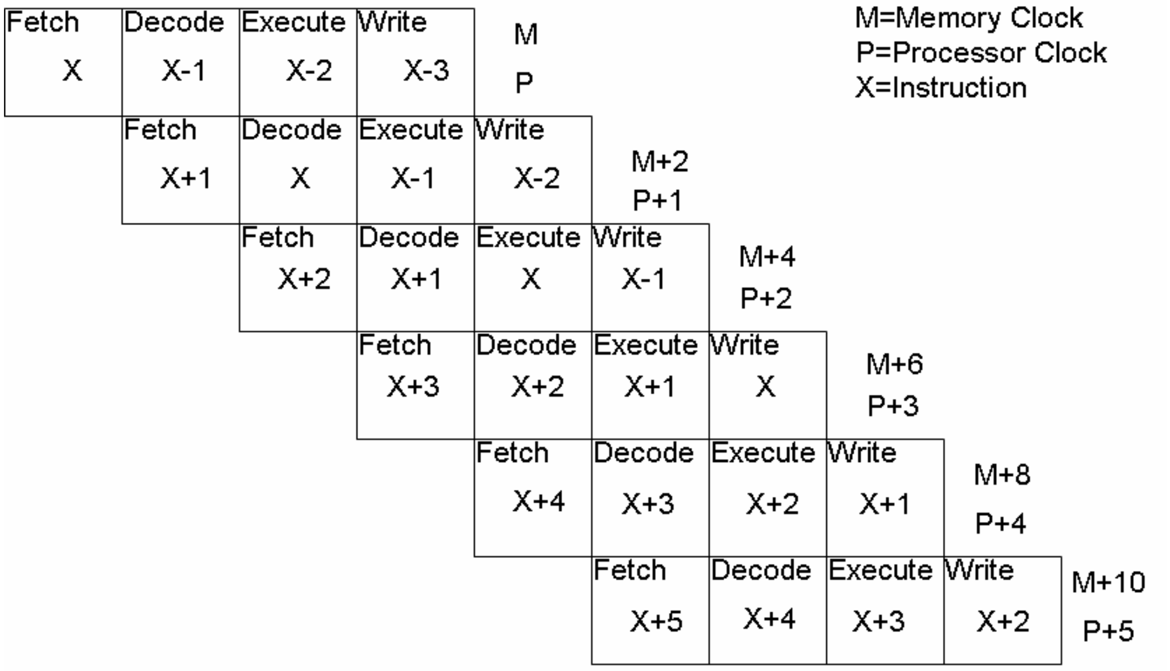

M=Memory Clock
P=Processor Clock
X=Instruction
Fetch X | Decode X-1 | Execute X-2 | Write X-3 | M P
Fetch X+1 | Decode X | Execute X-1 | Write X-2 | M+2 P+1
Fetch X+2 | Decode X+1 | Execute X | Write X-1 | M+4 P+2
Fetch X+3 | Decode X+2 | Execute X+1 | Write X | M+6 P+3
Fetch X+4 | Decode X+3 | Execute X+2 | Write X+1 | M+8 P+4
Fetch X+5 | Decode X+4 | Execute X+3 | Write X+2 | M+10 P+5

Figure 17. KDLX Pipeline with no Errors

When an error occurs, the pipeline is halted while the Error Syndrome is stored to memory. Once the pipeline starts back up, a TRAP instruction is generated. There was a concern that the instructions still in the pipeline could force another TRAP.

Figure 18 shows the KDLX pipeline with an error detected, the TRAP instruction inserted, and the first ISR instruction being fetched from memory. When an error is detected in instruction *X*–3, there are three additional instructions that will be executed prior to the TRAP instruction. If the error is in the program counter of a processor, each of these instructions will generate an error. This will halt the processors and save Error

Syndromes for each error. It will not inject another TRAP instruction, since the state machine is controlling the Error Save states.

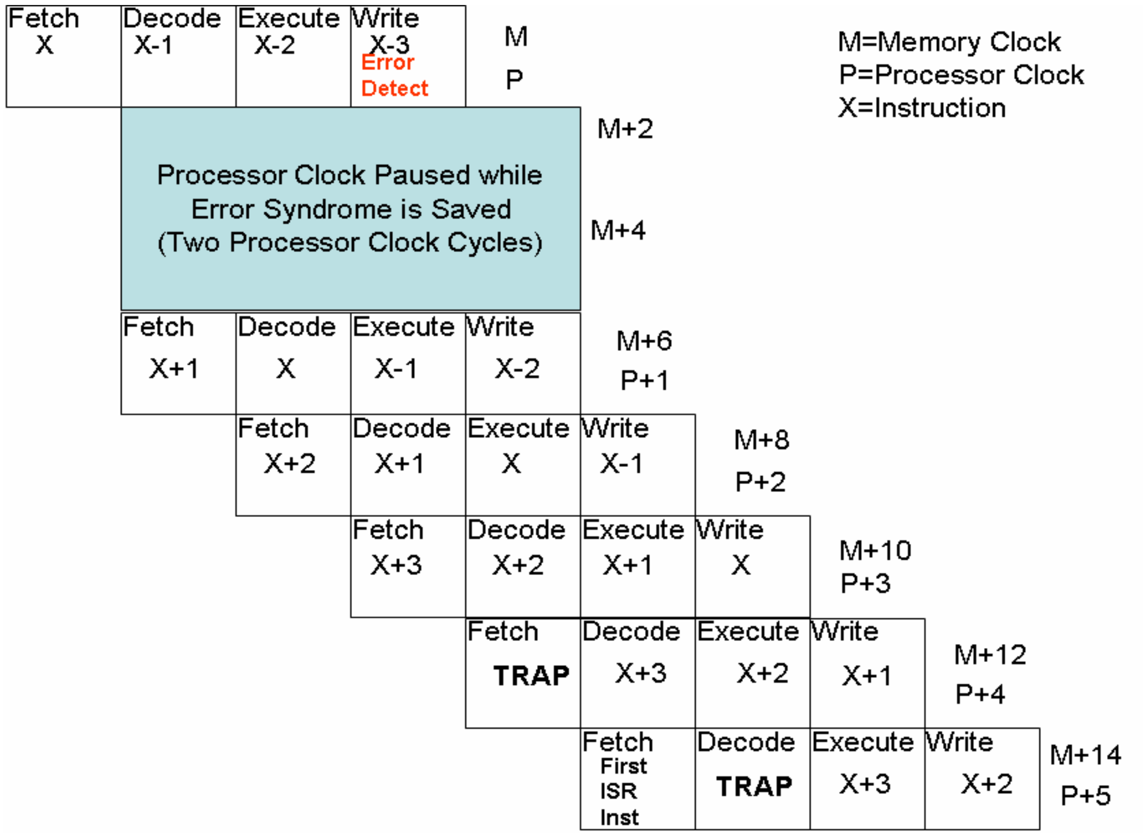


Figure 18. Pipeline with TRAP and ISR Instructions

F. CHAPTER SUMMARY

This Chapter described the major components of the CFTP, and their integration into a full design. The next Chapter will discuss conclusions and follow-up research.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND FOLLOW-ON RESEARCH

The intent of this thesis was to implement a configurable fault tolerant processor on a single chip. The research showed that it is possible to fit the entire design onto a FPGA. The research also showed that this FPGA has the capacity for a larger design, such as a more complex microprocessor than the 16-bit RISC KDLX. This extra capacity allows for flexibility once the CFTP is on orbit.

A. OVERVIEW

Prior to the start of this research, the intent was to use a soft-core processor that had a Von Neumann architecture. Since the research used a Harvard architecture, the *Reconciler* was built to coordinate memory accesses. In order to control the various conditions the CFTP would transition through, Normal, Error Save, Error Correct, the finite state machine *Interrup* was built. And to save the 104-bit Error Syndrome to memory on a 32-bit bus, the Error Syndrome Storage Device was designed. Each of these components was new to the design.

The TMR Assembly was modified from previous research. The single bit voter was adapted to only indicate error, and not differentiate between voter and data error. And redesigning the TMRA to accommodate the KDLX processor architecture was also a part of this research.

B. CONCLUSIONS

The CFTP is a viable design for space based application. It will detect and correct errors in the processor caused by SEUs. It also allows for on-orbit reconfiguration for both errors in design and future upgrade. The CFTP research helps continue to improve the capability of space based systems, bringing newer systems to space at a faster rate than currently capable.

C. FOLLOW-ON RESEARCH

There are several areas for further research. Implementation of this design in an FPGA is required, along with functional testing of the FPGA. This will ensure that the design created for this research functions properly in a real world environment.

Next, testing the design in a controlled radiation environment such as a cyclotron or other particle accelerator will be necessary before the device is launched. This will determine if the design is capable of handling the rigors of space instead of expending resources to launch a faulty design.

Investigation of a more complex processor to replace the KDLX processor should also be conducted. If possible, using a processor more closely representing current levels in state of the art of processor capability would be extremely beneficial to the space community.

Completing the interface between the CFTP and its off-chip components, such as the memory and off-board bus must be accomplished to allow for complete functionality. This includes an Error Correction and Detection (EDAC) component in order to prevent SEUs in memory from introducing errors into the system.

Finally, writing the program for the processor to run that best tests for errors has yet to be done.

APPENDIX A: CFTP SCHEMATICS AND CODE

Appendix A contains all the schematics and VHDL code files that were specifically built for this thesis. It does not contain any of the schematics contained in the Xilinx libraries, such as Flip-Flop or tri-state buffer schematics. Details of Xilinx schematics can be found in the Xilinx library files.

The schematic files in Appendix A include the full design schematic, and schematics of the sub-components *Reconciler*, Error Syndrome Storage Device (ESSD), Triple Modular Redundant Assembly (TMRA), the single bit voter, the 16-bit voter, and the 51-to-1 OR devices. The state machine diagram for the *Interrup* device is at the end of the schematic designs. It also includes the VHDL for all the associated KDLX files, as well as the state machine design from which the *Interrup* VHDL code was derived.

The VHDL files for the KDLX processor were not created specifically for this research [2]. But since KDLX processor is an integral part of this overall design, they are included.

A. SCHEMATICS

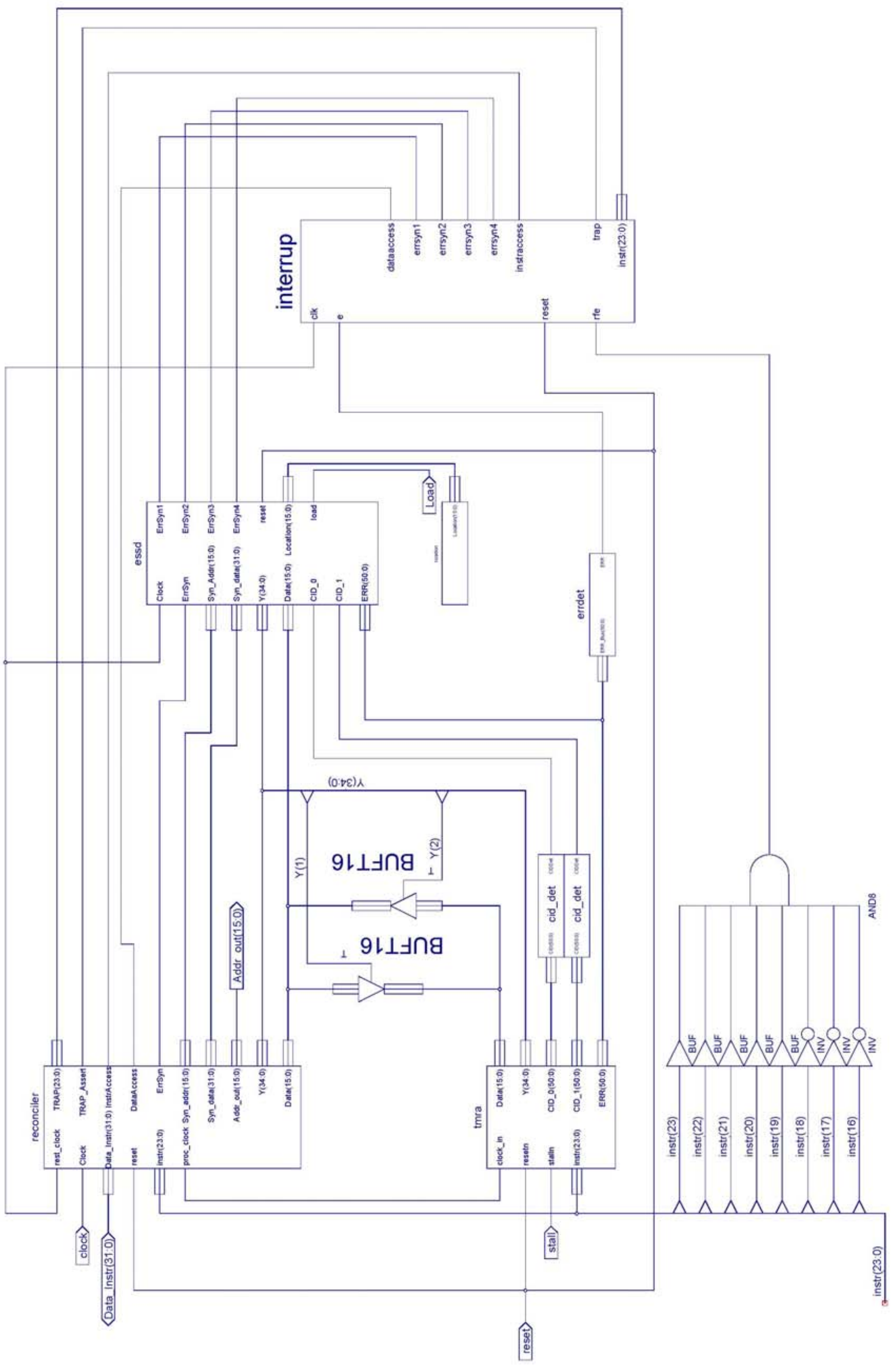


Figure 19. Full Design Schematic

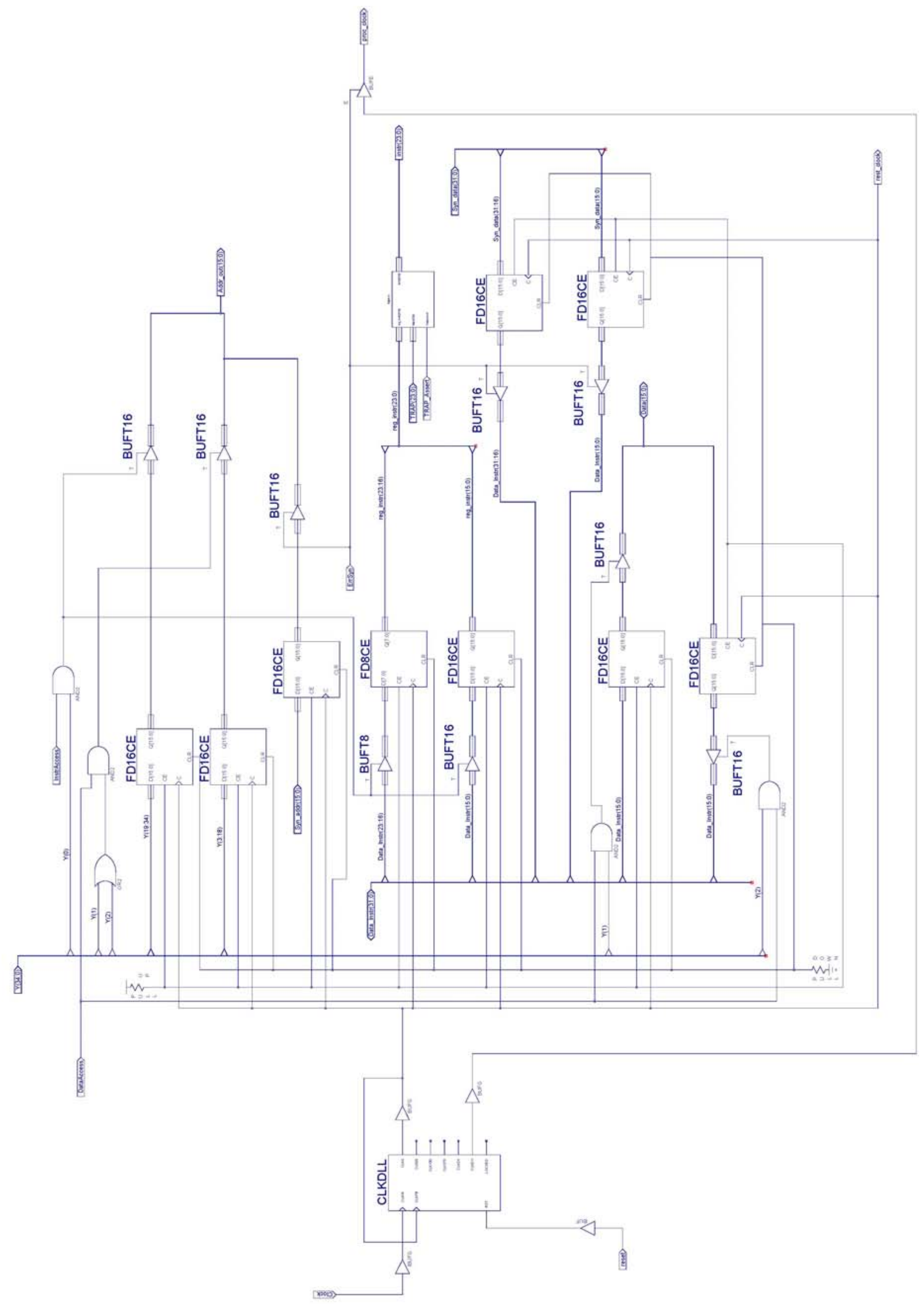


Figure 20. Reconciler

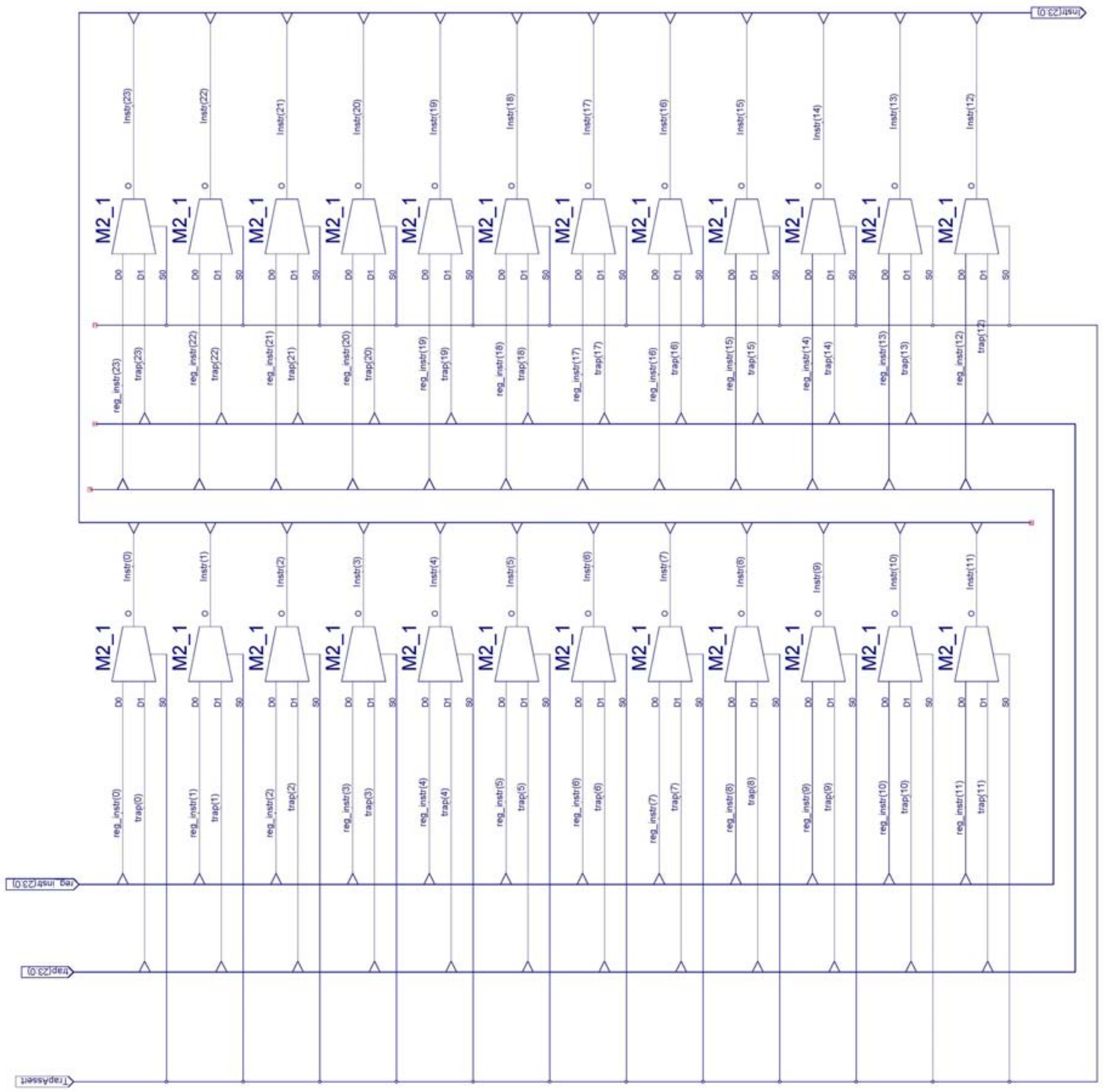


Figure 21.   24-bit 2-to-1 Multiplexer

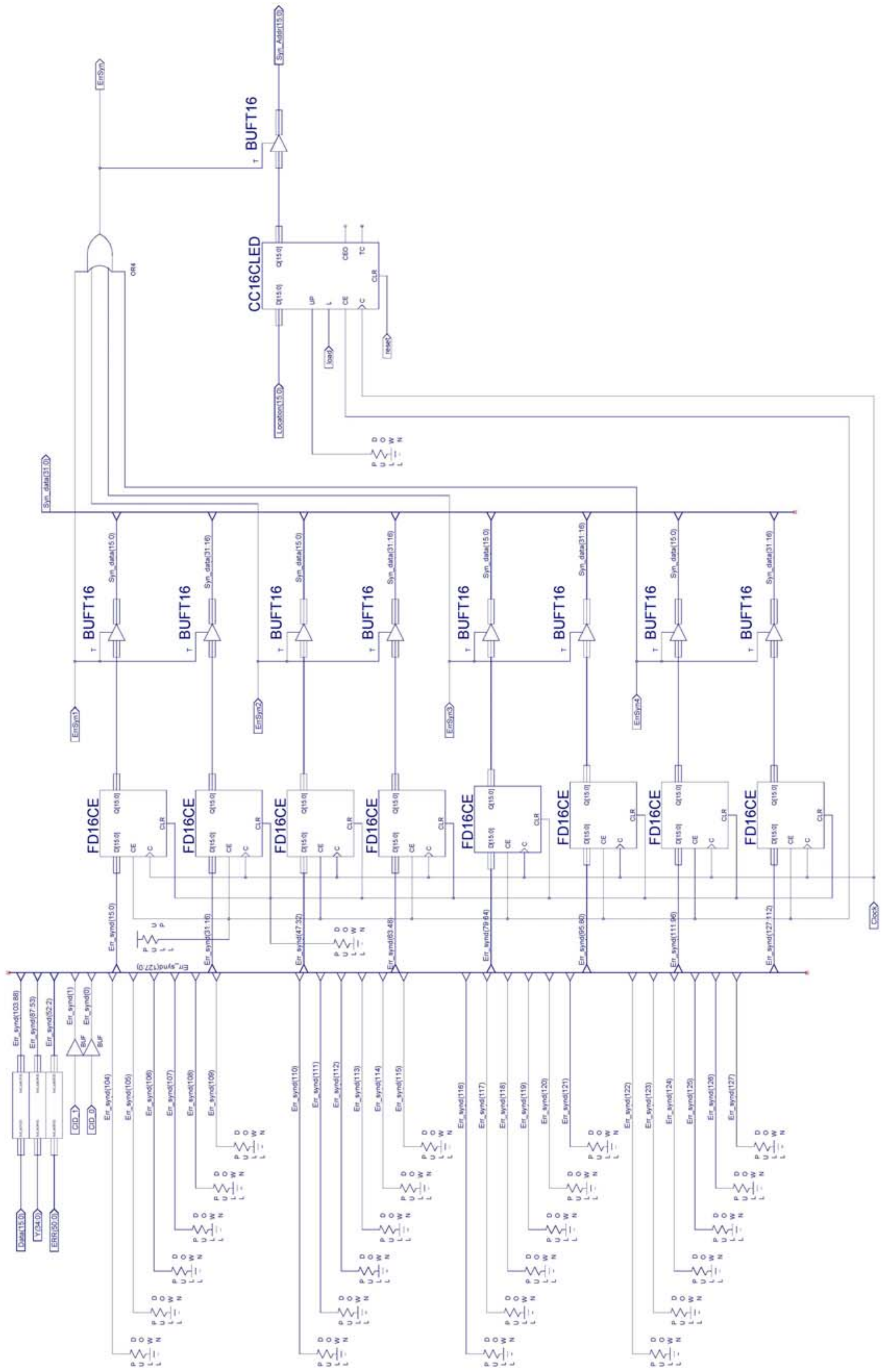


Figure 22. Error Syndrome Storage Device

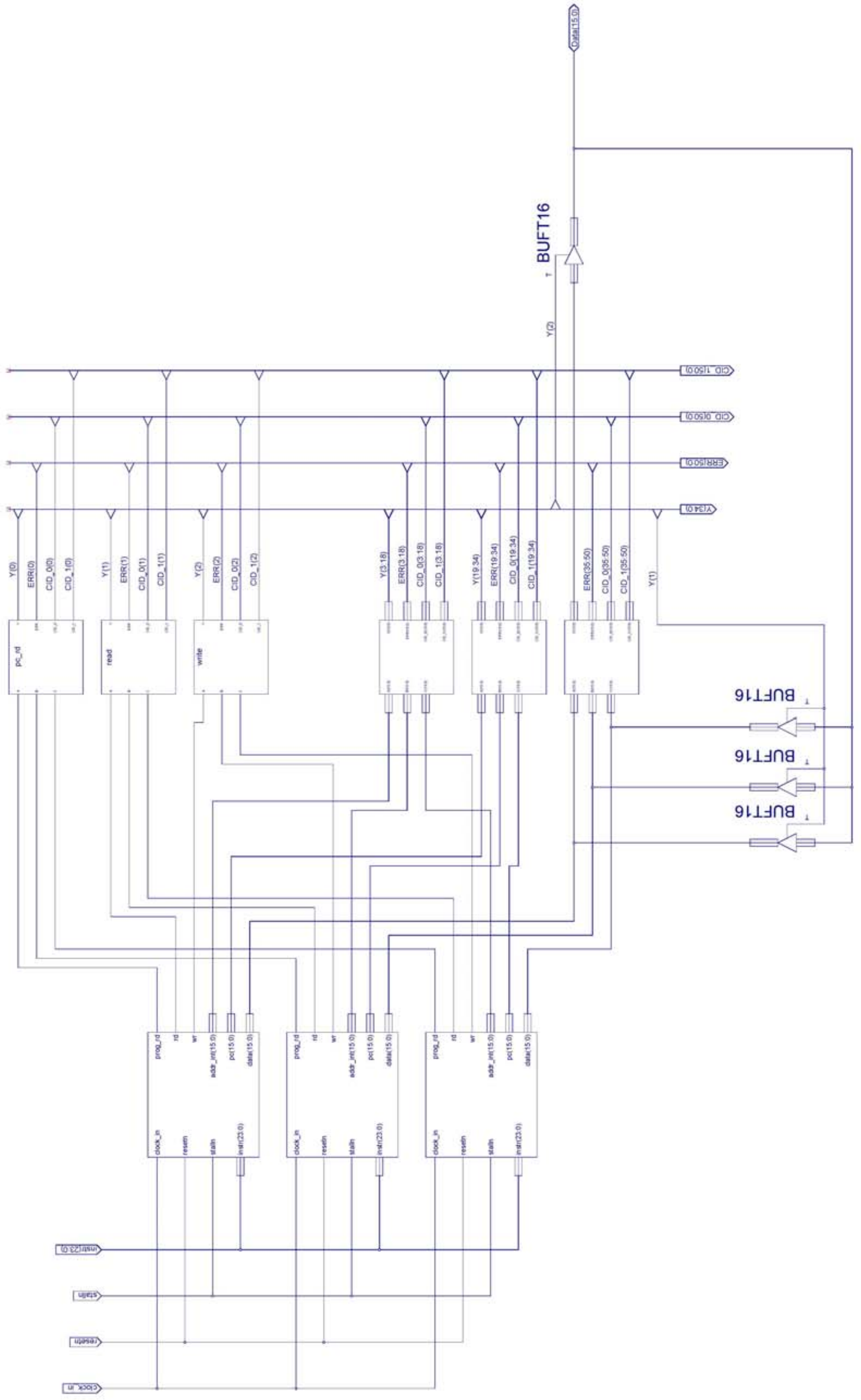


Figure 23. Triple Modular Redundant Assembly

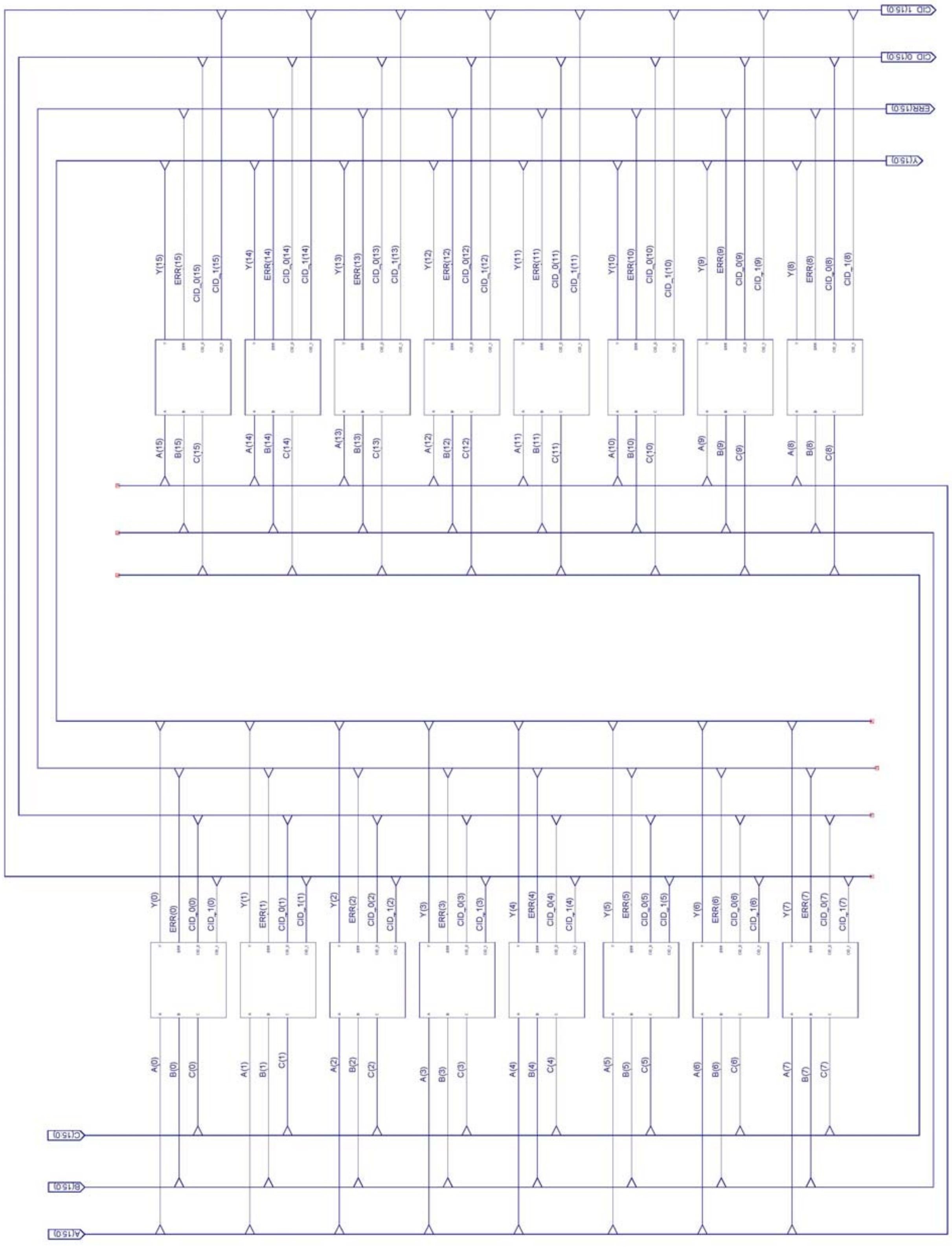


Figure 24. 16-Bit Voter

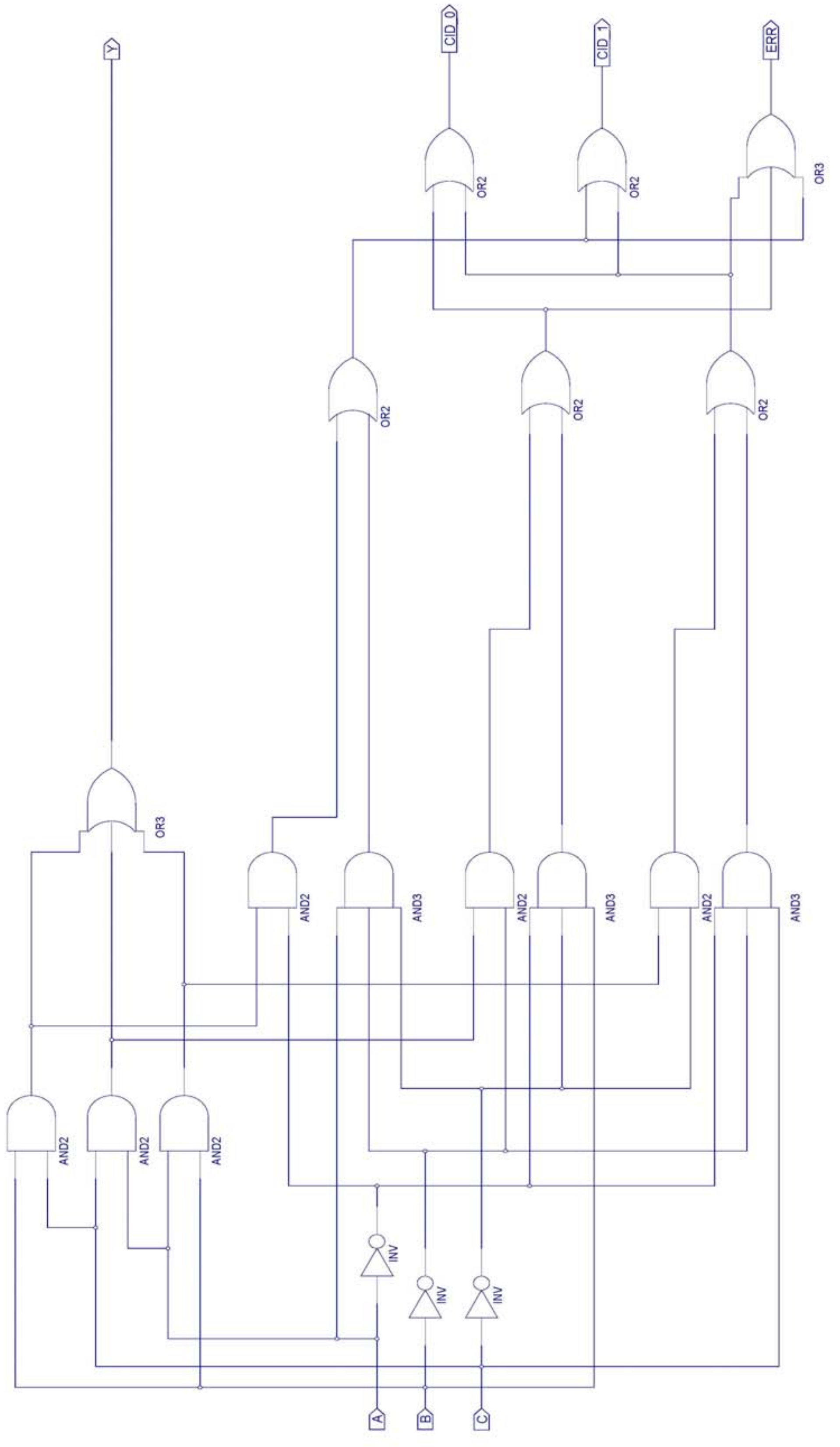


Figure 25. Single Bit Voter

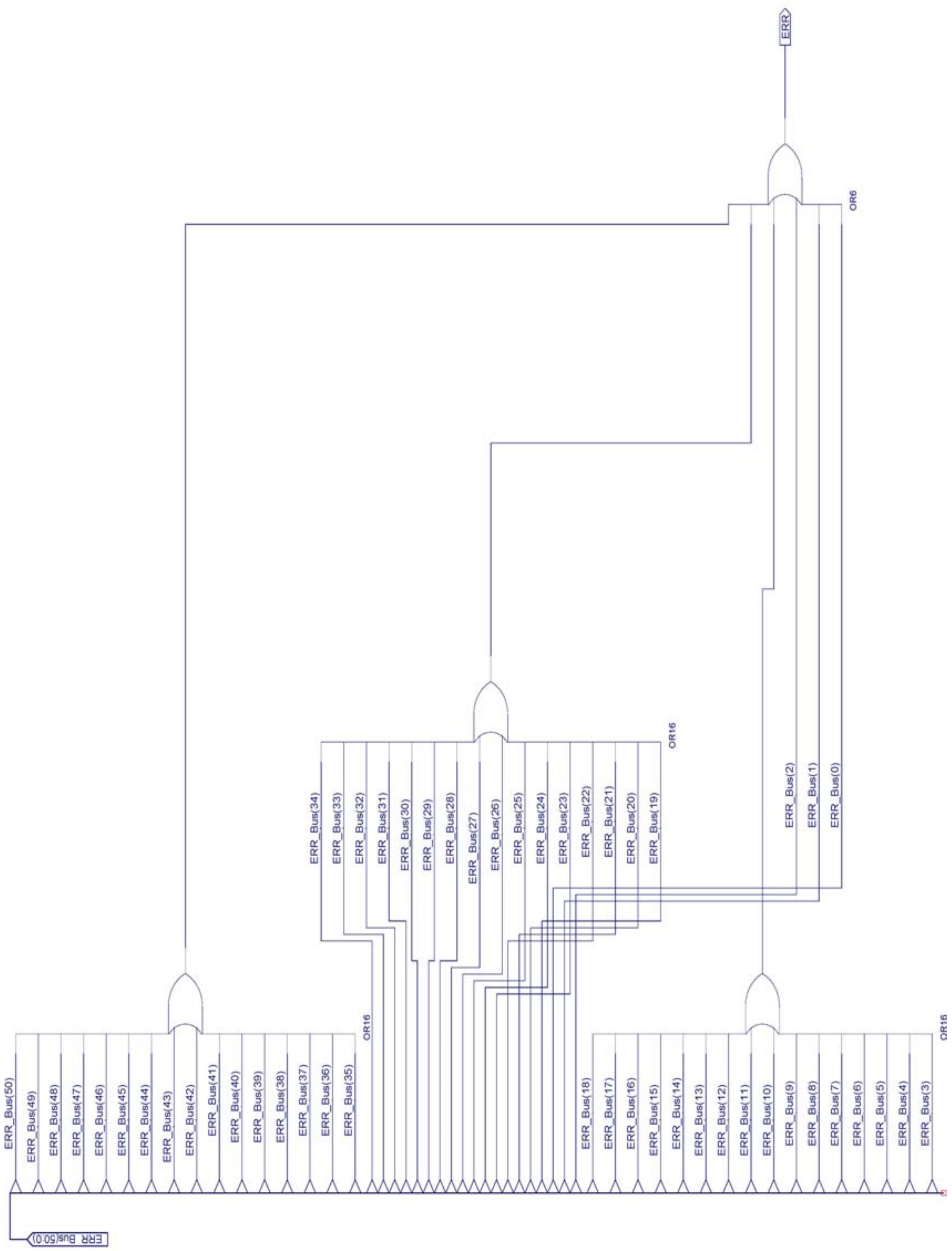


Figure 26. 51-to-1 OR Gate

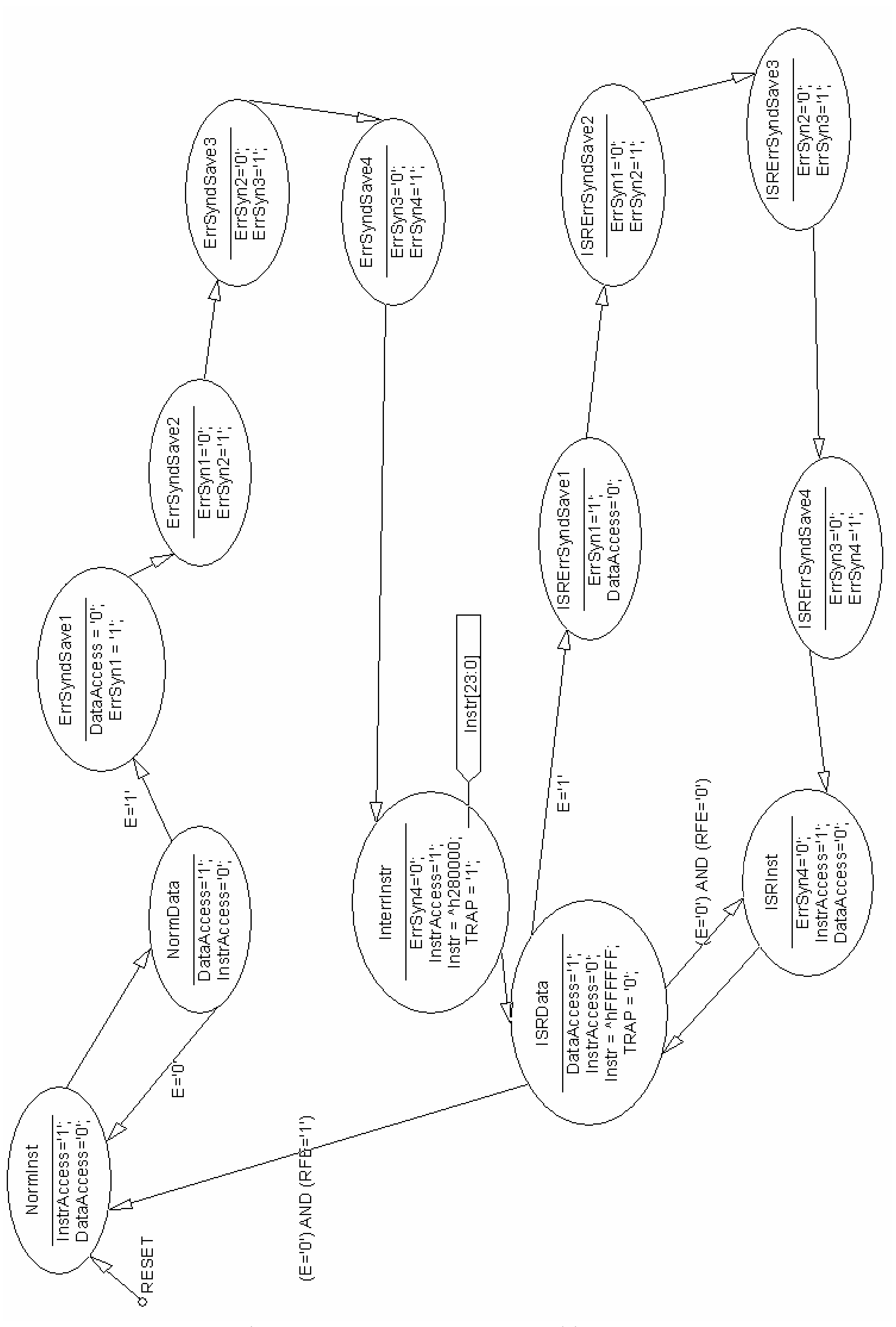

NormInst
InstrAccess='1'; DataAccess='0';
RESET
NormData
DataAccess='1'; InstrAccess='0';
E='1'
E='0'
ErrSyndSave1
DataAccess = '0'; ErrSyn1 = '1';
ErrSyndSave2
ErrSyn1='0'; ErrSyn2='1';
ErrSyndSave3
ErrSyn2='0'; ErrSyn3='1';
ErrSyndSave4
ErrSyn3='0'; ErrSyn4='1';
InterrInstr
ErrSyn4='0'; InstrAccess='1'; Instr = ^h280000; TRAP = '1';
Instr[23:0]
ISRData
DataAccess='1'; InstrAccess='0'; Instr = ^hFFFFFF; TRAP = '0';
(E='0') AND (RFE='1')
E='1'
ISRErrSyndSave1
ErrSyn1='1'; DataAccess='0';
ISRErrSyndSave2
ErrSyn1='0'; ErrSyn2='1';
ISRErrSyndSave3
ErrSyn2='0'; ErrSyn3='1';
ISRErrSyndSave4
ErrSyn3='0'; ErrSyn4='1';
(E='0') AND (RFE='0')
ISRInst
ErrSyn4='0'; InstrAccess='1'; DataAccess='0';

Figure 27. Interrup State Machine

B. VHDL FILES

1. adder.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;
USE IEEE.std_logic_unsigned.all;

-- ***** adder model *****
-- external ports
ENTITY adder IS PORT (
      A : IN std_logic_vector(15 downto 0);
            B:   IN std_logic_vector(15 downto 0);
      alu_op1 : IN std_logic;
      alu_op3 : IN std_logic;
      alu_op4 : IN std_logic;
      Out_word : OUT std_logic_vector(15 downto 0)
);
END adder;

-- internal structure
ARCHITECTURE rtl OF adder IS

-- COMPONENTS

COMPONENT AO22
PORT (
      A : IN std_logic;
      B : IN std_logic;
      C : IN std_logic;
      D : IN std_logic;
      zOut : OUT std_logic
);
END COMPONENT;

SIGNAL Vdd : std_logic;
SIGNAL subtract : std_logic;
-- INSTANCES
BEGIN
Vdd <= '1';
AO22_1 : AO22   PORT MAP(
      A => Vdd,
      B => alu_op1,
      C => alu_op4,
      D => alu_op3,
      zOut => subtract
);

process (A, B, subtract)
begin
  if (subtract = '1') then
   out_word <= A-B;
  else out_word <= A+B;
  end if;
end process;
END rtl;
```

2. alu.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

-- ***** alu model *****
-- external ports
ENTITY alu IS PORT (
      A : IN std_logic_vector (15 downto 0);
  alu_op : IN std_logic_vector (4 downto 0);
      alu_out : OUT std_logic_vector (15 downto 0);
      B : IN std_logic_vector (15 downto 0)
);
END alu;

-- internal structure
ARCHITECTURE structural OF alu IS

-- COMPONENTS

COMPONENT adder
PORT (
      A : IN std_logic_vector(15 downto 0);
      B : IN std_logic_vector (15 downto 0);
      alu_op1 : IN std_logic;
      alu_op3 : IN std_logic;
      alu_op4 : IN std_logic;
      Out_word : OUT std_logic_vector (15 downto 0)
);
END COMPONENT;

COMPONENT alu_logic
PORT (
      A : IN std_logic_vector (15 downto 0);
      B : IN std_logic_vector (15 downto 0);
      Func : IN std_logic_vector (1 downto 0);
      logic_out : OUT std_logic_vector (15 downto 0)
);
END COMPONENT;

COMPONENT log_barrel
PORT (
      ar_or_log : IN std_logic;
      In_Word : IN std_logic_vector (15 downto 0);
      l_or_r : IN std_logic;
      Out_word : OUT std_logic_vector (15 downto 0);
      Shift : IN std_logic_vector (3 downto 0)
);
END COMPONENT;

COMPONENT word_mux4
PORT (
      A : IN std_logic_vector (15 downto 0);
      B : IN std_logic_vector (15 downto 0);
      C : IN std_logic_vector (15 downto 0);
      D : IN std_logic_vector (15 downto 0);
  Sel : IN std_logic_vector (1 downto 0);
```

```
      Out_word : OUT std_logic_vector (15 downto 0)
);
END COMPONENT;

COMPONENT word_set
PORT (
      In_word : IN std_logic_vector (15 downto 0);
      set_op : IN std_logic_vector (2 downto 0);
      set_out : OUT std_logic
);
END COMPONENT;

-- SIGNALS

SIGNAL set_out : std_logic_vector (15 downto 0);
SIGNAL log_barrel_out : std_logic_vector (15 downto 0);
SIGNAL logic_out : std_logic_vector (15 downto 0);
SIGNAL Adder_Out : std_logic_vector (15 downto 0);

-- INSTANCES
BEGIN
set_out(15 downto 1) <= "000000000000000";
halfword_adder_1 : adder   PORT MAP(
      A => A,
      alu_op1 => alu_op(1),
      alu_op3 => alu_op(3),
      alu_op4 => alu_op(4),
      B => B,
      Out_word  => Adder_Out
);
halfword_alu_logic_1 : alu_logic   PORT MAP(
      A => A,
      B => B,
      Func => alu_op(1 downto 0),
      logic_out => logic_out
);
halfword_log_barrel_1 : log_barrel   PORT MAP(
      ar_or_log => alu_op(0),
      In_word => A,
      l_or_r => alu_op(1),
        Out_word => log_barrel_out,
      Shift => B(3 downto 0)
);
halfword_mux4_1 : word_mux4   PORT MAP(
      A => Adder_Out,
      B => logic_out,
      C => log_barrel_out,
      D => set_out,
      Out_word => alu_out,
      Sel => alu_op(4 downto 3)
);
halfword_set_1 : word_set   PORT MAP(
      In_word => Adder_Out,
      set_op => alu_op(2 downto 0),
      set_out => set_out(0)
);
END structural;
```

3. alu_logic.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

-- ***** alu_logic model *****
-- external ports
ENTITY alu_logic IS PORT (
      A: IN std_logic_vector(15 downto 0);
      B : IN std_logic_vector(15 downto 0);
      Func: IN std_logic_vector(1 downto 0);
      logic_out : OUT std_logic_vector(15 downto 0)
);
END alu_logic;

-- internal structure
ARCHITECTURE rtl OF alu_logic IS

BEGIN

process (A,B, func)
begin
  case func is
   when "00" => logic_out <= A;
   when "01" => logic_out <= (A and B);
   when "10" => logic_out <= (A or B);
   when others => logic_out <= (A xor B);
   end case;
end process;


END rtl;
```

4. AO22.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;


entity AO22 is port (
  A, B, C, D: IN std_logic;
  zOut : OUT std_logic);
end AO22;

architecture behavioral of AO22 is
begin
  zOut <= (A and B) or (C and D);
end behavioral;
```

5. core.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;
-- ***** core model *****
-- external ports
ENTITY core IS PORT (
      Addr_Int : OUT std_logic_vector(15 downto 0);
      Clock_in : IN std_logic;
      Input_Data : IN std_logic_vector(15 downto 0);
  Output_Data : Out std_logic_vector(15 downto 0);
      Instr : IN std_logic_vector(23 downto 0);
      PC : OUT std_logic_vector(15 downto 0);
      Prog_Rd : OUT std_logic;
      Rd : OUT std_logic;
      Resetn : IN std_logic;
      Stalln : IN std_logic;
      Wr : OUT std_logic
);
END core;

-- internal structure
ARCHITECTURE structural OF core IS

-- COMPONENTS

COMPONENT alu
PORT (
      A : IN std_logic_vector(15 downto 0);
      alu_op : IN std_logic_vector(4 downto 0);
      alu_out : OUT std_logic_vector(15 downto 0);
      B : IN std_logic_vector(15 downto 0)
);
END COMPONENT;


COMPONENT word_mux3
PORT (
      A : IN std_logic_vector(15 downto 0);
      B : IN std_logic_vector(15 downto 0);
      C : IN std_logic_vector(15 downto 0);
      Out_word : OUT std_logic_vector(15 downto 0);
      Sel : IN std_logic_vector(1 downto 0)
);
END COMPONENT;

COMPONENT word_mux4
PORT (
      A : IN std_logic_vector(15 downto 0);
      B : IN std_logic_vector(15 downto 0);
      C : IN std_logic_vector(15 downto 0);
      D : IN std_logic_vector(15 downto 0);
      Out_word : OUT std_logic_vector(15 downto 0);
      Sel : IN std_logic_vector(1 downto 0)
);
END COMPONENT;

COMPONENT regfile
PORT (
      A : OUT std_logic_vector(15 downto 0);
      B : OUT std_logic_vector(15 downto 0);
      clock : IN std_logic;
      Data_In : IN std_logic_vector(15 downto 0);
      Dest : IN std_logic_vector(3 downto 0);
        stalln: IN std_logic;
      resetn : IN std_logic;
      RSone : IN std_logic_vector(3 downto 0);
      RStwo : IN std_logic_vector(3 downto 0);
      scan_data_in : IN std_logic;
      scan_enable : IN std_logic;
      wb_enable : IN std_logic
);
END COMPONENT;

COMPONENT word_reg_single
PORT (
      Clock : IN std_logic;
      Data_In : IN std_logic_vector(15 downto 0);
      Data_out : OUT std_logic_vector(15 downto 0);
      Enable : IN std_logic;
      Resetn : IN std_logic;
      Scan_Data_In : IN std_logic;
      Scan_Enable : IN std_logic
);
END COMPONENT;

COMPONENT pc_control
PORT (
      ALU_Out : IN std_logic_vector(15 downto 0);
      Clock : IN std_logic;
      D2_Inc_PC : OUT std_logic_vector(15 downto 0);
        D_Link_PC : OUT std_logic_vector(15 downto 0);
      IAR_Enable : IN std_logic;
      PC : OUT std_logic_vector(15 downto 0);
      PC_Sel : IN std_logic_vector(1 downto 0);
      Resetn : IN std_logic;
      Scan_Data_In : IN std_logic;
      Scan_Data_Out : OUT std_logic;
      Scan_Enable : IN std_logic;
      Stalln : IN std_logic
);
END COMPONENT;

COMPONENT pipeline
PORT (
alu_op : OUT std_logic_vector(4 downto 0);
        A_Mux : OUT std_logic_vector(1 downto 0);
        B_Mux : OUT std_logic_vector(1 downto 0);
        Clock : IN std_logic;
        Data_In : IN std_logic_vector(23 downto 0);
        Dest : OUT std_logic_vector(3 downto 0);
        Immed : OUT std_logic_vector(15 downto 0);
        PC_Sel : OUT std_logic_vector(1 downto 0);
        rd_enable : OUT std_logic;
        Reg_In_Sel : OUT std_logic_vector(1 downto 0);
        Resetn : IN std_logic;
        RSone : OUT std_logic_vector(3 downto 0);
        RStwo : OUT std_logic_vector(3 downto 0);
        Scan_Data_In : IN std_logic;
        Scan_Enable : IN std_logic;
        Stalln : IN std_logic;
        wb_enable : OUT std_logic;
        scan_out : OUT std_logic;
        IAR_Enable : OUT std_logic;
        wr_enable : OUT std_logic;
        zero_flag : IN std_logic
);
END COMPONENT;

COMPONENT rw_control
PORT (
Clock : IN std_logic;
        Prog_Rd : OUT std_logic;
        Rd : OUT std_logic;
        rd_enable : IN std_logic;
        resetn : IN std_logic;
        stalln : IN std_logic;
        Wr : OUT std_logic;
        wr_enable : IN std_logic
);
END COMPONENT;


COMPONENT zero_test
PORT (
      In_word : IN std_logic_vector(15 downto 0);
      zero_flag : OUT std_logic
);
END COMPONENT;

-- SIGNALS

SIGNAL wr_enable : std_logic;
SIGNAL zero_flag : std_logic;
SIGNAL IAR_Enable : std_logic;
SIGNAL wb_enable : std_logic;
SIGNAL pipeline_scan_out : std_logic;
SIGNAL Dest : std_logic_vector(3 downto 0);
SIGNAL A : std_logic_vector(15 downto 0);
SIGNAL D2_Inc_PC : std_logic_vector(15 downto 0);
SIGNAL Immed : std_logic_vector(15 downto 0);
SIGNAL D_ALU_Out : std_logic_vector(15 downto 0);
SIGNAL D_Link_PC : std_logic_vector(15 downto 0);
SIGNAL Reg_In_Sel : std_logic_vector(1 downto 0);
SIGNAL ALU_A : std_logic_vector(15 downto 0);
SIGNAL ALU_Out : std_logic_vector(15 downto 0);
SIGNAL ALU_B : std_logic_vector(15 downto 0);
SIGNAL Gnd : std_logic;
SIGNAL B : std_logic_vector(15 downto 0);
SIGNAL LD_Memory_In : std_logic_vector(15 downto 0);
SIGNAL output_en_n : std_logic;
SIGNAL rd_enable : std_logic;
SIGNAL pc_control_scan_out : std_logic;
SIGNAL Buf_Stalln : std_logic;
SIGNAL Buf_resetn : std_logic;
SIGNAL Clock : std_logic;
```

```
      SIGNAL Buf_Addr_Int : std_logic_vector(15 downto 0);
      SIGNAL Shift_En : std_logic;
      SIGNAL alu_op : std_logic_vector(4 downto 0);
      SIGNAL Buf_Scan_Data_Out : std_logic;
      SIGNAL A_Mux : std_logic_vector(1 downto 0);
      SIGNAL B_Mux : std_logic_vector(1 downto 0);
      SIGNAL RSone : std_logic_vector(3 downto 0);
      SIGNAL RStwo : std_logic_vector(3 downto 0);
      SIGNAL PC_Sel : std_logic_vector(1 downto 0);
      SIGNAL Data_Out : std_logic_vector(15 downto 0);
      SIGNAL Regfile_In : std_logic_vector(15 downto 0);
      SIGNAL zero_byte : std_logic_vector(7 downto 0);
      SIGNAL Data_In : std_logic_vector(15 downto 0);
      SIGNAL sign_ext_immed : std_logic_vector(15 downto 0);
      SIGNAL scan_data_in : std_logic;
      -- INSTANCES
      BEGIN
      clock <= clock_in;
      shift_en <= '0';
      scan_data_in <= '0';
      Addr_Int <= Buf_Addr_Int;
      zero_byte <= "00000000";
      sign_ext_immed(15 downto 8) <= Immed(7) & Immed(7) & Immed(7) &
Immed(7) & Immed(7) & Immed(7) & Immed(7) & Immed(7);
      sign_ext_immed (7 downto 0) <= Immed(7 downto 0);
      Wr <= output_en_n;
      Output_Data <= Data_Out;

      Word_Reg_1 : word_reg_single PORT MAP(
            Clock => Clock,
            Data_In => B,
            Data_out => Data_Out,
            Enable => Stalln,
            Resetn => Resetn,
            Scan_Data_In => pc_control_scan_out,
            Scan_Enable => Shift_En
      );

      Word_Reg_2 : word_reg_single PORT MAP(
            Clock => Clock,
            Data_In => Input_Data,
            Data_out => LD_Memory_In,
            Enable => Stalln,
            Resetn => Resetn,
            Scan_Data_In => Data_Out(15),
            Scan_Enable => Shift_En
      );



      alu_1 : alu   PORT MAP(
            A => ALU_A,
            alu_op => alu_op,
            alu_out => ALU_Out,
            B => ALU_B
      );
```

```
word_mux3_1 : word_mux3   PORT MAP(
      A => D_ALU_Out,
      B => LD_Memory_In,
      C => D_Link_PC,
      Out_word => Regfile_In,
      Sel => Reg_In_Sel
);
word_mux3_2 : word_mux3   PORT MAP(
      A => B,
      B(7 downto 0) => Immed(7 downto 0),
      B(15 downto 8) => zero_byte,
      C => sign_ext_immed,
      Out_word => ALU_B,
      Sel => B_Mux
);

word_mux4_1 : word_mux4   PORT MAP(
      A => A,
      B => D2_Inc_PC,
      C(7 downto 0) => zero_byte,
      C(15 downto 8) => Immed(7 downto 0),
      D => Immed(15 downto 0),
      Out_word => ALU_A,
      Sel => A_Mux
);

regfile_1 : regfile   PORT MAP(
      A => A,
      B => B,
      clock => Clock,
      Data_In => regfile_in,
      Dest => Dest,
        stalln => stalln,
      resetn => resetn,
      RSone => RSone,
      RStwo => RStwo,
      scan_data_in => pipeline_scan_out,
      scan_enable => Shift_En,
      wb_enable => wb_enable
);

word_reg_single_3 : word_reg_single   PORT MAP(
      Clock => Clock,
      Data_In => Buf_Addr_Int,
      Data_out => D_ALU_Out,
      Enable => Stalln,
      Resetn => resetn,
      Scan_Data_In => Buf_Addr_Int(15),
      Scan_Enable => Shift_En
);

word_reg_single_4 : word_reg_single   PORT MAP(
      Clock => Clock,
      Data_In => ALU_Out,
      Data_out => Buf_Addr_Int,
      Enable => Stalln,
      Resetn => resetn,
      Scan_Data_In => B(15),
      Scan_Enable => Shift_En
);
pc_control_1 : pc_control   PORT MAP(
      ALU_Out => ALU_Out,
      Clock => Clock,
      D2_Inc_PC => D2_Inc_PC,
      D_Link_PC => D_Link_PC,
      IAR_Enable => IAR_Enable,
      PC => PC,
      PC_Sel => PC_Sel,
      Resetn => resetn,
      Scan_Data_In => D_ALU_Out(15),
      Scan_Data_Out => pc_control_scan_out,
      Scan_Enable => Shift_En,
      Stalln => Stalln
);
pipeline_1 : pipeline   PORT MAP(
      alu_op => alu_op,
      A_Mux => A_Mux,
      B_Mux => B_Mux,
      Clock => Clock,
      Data_In => Instr,
      Dest => Dest,
      Immed => Immed,
      PC_Sel => PC_Sel,
      rd_enable => rd_enable,
      Reg_In_Sel => Reg_In_Sel,
      Resetn => resetn,
      RSone => RSone,
      RStwo => RStwo,
      Scan_Data_In => Scan_Data_In,
      Scan_Enable => Shift_En,
      Stalln => Stalln,
      wb_enable => wb_enable,
      scan_out => pipeline_scan_out,
      IAR_Enable => IAR_Enable,
      wr_enable => wr_enable,
      zero_flag => zero_flag
);
rw_control_1 : rw_control   PORT MAP(
      Clock => Clock,
      Prog_Rd => Prog_Rd,
      Rd => Rd,
      rd_enable => rd_enable,
      resetn => resetn,
      stalln => Stalln,
      Wr => output_en_n,
      wr_enable => wr_enable
);
zero_test_1 : zero_test   PORT MAP(
      In_word => A,
      zero_flag => zero_flag
);
END structural;
```

6. Dest_Decoder.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

-- ***** Dest_Decoder model *****
-- external ports
ENTITY Dest_Decoder IS PORT (
      Dest : IN std_logic_vector(3 downto 0);
      Enable : OUT std_logic_vector(15 downto 1);
      wb_enable : IN std_logic
);
END Dest_Decoder;

-- internal structure
ARCHITECTURE rtl OF Dest_Decoder IS


-- SIGNALS

SIGNAL buf_enable : std_logic_vector(15 downto 1);

-- INSTANCES
BEGIN
with dest select
buf_enable <=  "000000000000001" when  "0001",
                       "000000000000010" when  "0010",
                       "000000000000100" when  "0011",
                       "000000000001000" when  "0100",
                       "000000000010000" when  "0101",
                       "000000000100000" when  "0110",
                       "000000001000000" when  "0111",
                       "000000010000000" when  "1000",
                       "000000100000000" when  "1001",
                       "000001000000000" when  "1010",
                       "000010000000000" when  "1011",
                       "000100000000000" when  "1100",
                       "001000000000000" when  "1101",
                       "010000000000000" when  "1110",
                       "100000000000000" when others;

      Enable <= buf_enable when (wb_enable = '1') else
"000000000000000";
      END rtl;
```

7. dlx.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;
-- ***** dlx model *****
-- external ports
ENTITY dlx IS PORT (
      Addr_Int : OUT std_logic_vector(15 downto 0);
      Clock_in : IN std_logic;
      Data : INOUT std_logic_vector(15 downto 0);
      Instr : IN std_logic_vector(23 downto 0);
      PC : OUT std_logic_vector(15 downto 0);
      Prog_Rd : OUT std_logic;
      Rd : OUT std_logic;
      Resetn : IN std_logic;
      Stalln : IN std_logic;
      Wr : OUT std_logic
);

END dlx;


-- internal structure
ARCHITECTURE structural OF dlx IS


-- COMPONENTS


COMPONENT core
PORT (
      Addr_Int : OUT std_logic_vector(15 downto 0);
      Clock_in : IN std_logic;
      Input_Data : IN std_logic_vector(15 downto 0);
      Output_Data : Out std_logic_vector(15 downto 0);
      Instr : IN std_logic_vector(23 downto 0);
      PC : OUT std_logic_vector(15 downto 0);
      Prog_Rd : OUT std_logic;
      Rd : OUT std_logic;
      Resetn : IN std_logic;
      Stalln : IN std_logic;
      Wr : OUT std_logic
);

END COMPONENT;

COMPONENT IO_Pads
PORT (
      Pads : INOUT std_logic_vector (15 downto 0);
      In_Data : Out std_logic_vector (15 downto 0);
    Out_Data : In std_logic_vector (15 downto 0);
    Output_En_n : IN std_logic
);
END COMPONENT;




-- SIGNALS
signal Input_data : std_logic_vector(15 downto 0);
signal Output_data : std_logic_vector(15 downto 0);
signal wr_int : std_logic;

-- INSTANCES
BEGIN
wr <= wr_int;
core1 : core PORT MAP(
      Addr_Int => Addr_Int,
      Clock_in => Clock_In,
      Input_Data => Input_data,
      Output_Data => Output_data,
      Instr => Instr,
      PC => PC,
      Prog_Rd => Prog_Rd,
      Rd => Rd,
      Resetn => Resetn,
      Stalln => stalln,
      Wr => Wr_int
);

IO_Pads_1 : IO_Pads PORT MAP(
    Pads => Data,
    In_Data => Input_Data,
    Out_Data => Output_Data,
    Output_En_n => wr_int
);


END structural;
```

8. increment.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;
USE IEEE.std_logic_unsigned.all;

-- ***** increment model *****
-- external ports
ENTITY increment IS PORT (
      CI : IN std_logic;
      In_word : IN std_logic_vector (15 downto 0);
      Out_word : OUT std_logic_vector (15 downto 0)
);
END increment;

-- rtl
ARCHITECTURE rtl OF increment IS

BEGIN
process (In_word, CI)
begin
  Out_word <= In_word + CI;
end process;
END rtl;
```

9. IO_pads.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

---- *** IO_Pads Model ***
```

```
---- external ports

Entity IO_Pads is PORT (
      Pads : INOUT std_logic_vector (15 downto 0);
      In_Data : Out std_logic_vector (15 downto 0);
    Out_Data : In std_logic_vector (15 downto 0);
    Output_En_n : IN std_logic
);
END IO_Pads;

Architecture Behavior of IO_Pads is
Begin
  In_Data <= Pads;
  Pads <= Out_Data when Output_En_n = '0' else (Pads'range =>
'Z');
end Behavior;
```

10. log_barrel.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

-- ***** log_barrel model *****
-- external ports
ENTITY log_barrel IS PORT (
      ar_or_log : IN std_logic;
      In_word : IN std_logic_vector(15 downto 0);
      l_or_r : IN std_logic;
      Out_word : Out  std_logic_vector(15 downto 0);
      Shift: IN std_logic_vector(3 downto 0)

);
END log_barrel;

-- internal structure
ARCHITECTURE rtl OF log_barrel IS

signal sel1, sel2, sel3, sel4 : std_logic_vector ( 1 downto 0);
signal buf0b, buf0c, buf0d : std_logic_vector (15 downto 0);
signal buf1a, buf1b, buf1c, buf1d : std_logic_vector (15 downto
0);
signal buf2a, buf2b, buf2c, buf2d : std_logic_vector (15 downto
0);
signal buf3a, buf3b, buf3c, buf3d : std_logic_vector (15 downto
0);

component word_mux4
port (a : in std_logic_vector (15 downto 0);
         b : in std_logic_vector (15 downto 0);
         c : in std_logic_vector (15 downto 0);
         d : in std_logic_vector (15 downto 0);
        sel :  in std_logic_vector (1 downto 0);
        out_word : out std_logic_vector (15 downto 0)
);
end component;
```

```
      begin
      sel1(1) <= l_or_r and shift(0);
      sel1(0) <= ar_or_log and shift(0);

      sel2(1) <= l_or_r and shift(1);
      sel2(0) <= ar_or_log and shift(1);

      sel3(1) <= l_or_r and shift(2);
      sel3(0) <= ar_or_log and shift(2);

      sel4(1) <= l_or_r and shift(3);
      sel4(0) <= ar_or_log and shift(3);


      buf0b <= in_word(14 downto 0) & "0";
      buf0c <= "0" & in_word(15 downto 1);
      buf0d <= in_word(15) & in_word(15 downto 1);

      buf1b <= buf1a(13 downto 0) & "00";
      buf1c <= "00" & buf1a(15 downto 2);
      buf1d <= buf1a(15) & buf1a(15) & buf1a(15 downto 2);

      buf2b <= buf2a(11 downto 0) & "0000";
      buf2c <= "0000" & buf2a(15 downto 4);
      buf2d <= buf2a(15) & buf2a(15) & buf2a(15) & buf2a(15) & buf2a(15
downto 4);

      buf3b <= buf3a(7 downto 0) & "00000000";
      buf3c <= "00000000" & buf3a(15 downto 8);
      buf3d <= buf3a(15) & buf3a(15) & buf3a(15) & buf3a(15) &
buf3a(15) & buf3a(15) & buf3a(15) & buf3a(15) & buf3a(15 downto 8);



      mux1:  word_mux4
      port map (
          a => in_word,
          b => buf0b,
          c => buf0c,
          d => buf0d,
          sel => sel1,
          out_word => buf1a
          );

      mux2:  word_mux4
      port map (
          a => buf1a,
          b => buf1b,
          c => buf1c,
          d => buf1d,
          sel => sel2,
          out_word => buf2a
          );

      mux3:  word_mux4
      port map (
          a => buf2a,
          b => buf2b,
          c => buf2c,
          d => buf2d,
          sel => sel3,
          out_word => buf3a
          );

      mux4:  word_mux4
      port map (
          a => buf3a,
          b => buf3b,
          c => buf3c,
          d => buf3d,
          sel => sel4,
          out_word => out_word);

      end rtl;
```

11. pc_control.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;


-- ***** pc_control model *****
-- external ports
ENTITY pc_control IS PORT (
      ALU_Out : IN std_logic_vector(15 downto 0);
      Clock : IN std_logic;
      D2_Inc_PC : OUT std_logic_vector(15 downto 0);
      D_Link_PC : OUT std_logic_vector(15 downto 0);
      IAR_Enable : IN std_logic;
      In_PC : OUT std_logic_vector(15 downto 0);
      PC : OUT std_logic_vector(15 downto 0);
      PC_Sel : IN std_logic_vector(1 downto 0);
      Resetn : IN std_logic;
      Scan_Data_In : IN std_logic;
      Scan_Data_Out : OUT std_logic;
      Scan_Enable : IN std_logic;
      Stalln : IN std_logic
);
END pc_control;

-- internal structure
ARCHITECTURE structural OF pc_control IS

-- COMPONENTS

COMPONENT word_reg_single
PORT (
      Clock : IN std_logic;
      Data_In : IN std_logic_vector(15 downto 0);
      Data_out : OUT std_logic_vector(15 downto 0);
      Enable : IN std_logic;
      Resetn : IN std_logic;
      Scan_Data_In : IN std_logic;
      Scan_Enable : IN std_logic
);
END COMPONENT;

COMPONENT word_mux3
PORT (
      A : IN std_logic_vector(15 downto 0);
      B : IN std_logic_vector(15 downto 0);
      C : IN std_logic_vector(15 downto 0);
      Out_word : OUT std_logic_vector(15 downto 0);
      Sel : IN std_logic_vector(1 downto 0)
);
END COMPONENT;

COMPONENT increment
PORT (
      CI : IN std_logic;
      In_word : IN std_logic_vector(15 downto 0);
      Out_word : OUT std_logic_vector(15 downto 0)
);
END COMPONENT;

-- SIGNALS

SIGNAL IAR : std_logic_vector(15 downto 0);
SIGNAL PC_Incr : std_logic_vector(15 downto 0);
SIGNAL Buf_In_PC : std_logic_vector(15 downto 0);
SIGNAL Buf_PC : std_logic_vector(15 downto 0);
SIGNAL Buf_Scan_Data_Out : std_logic;
SIGNAL Buf_D1_Inc_PC  : std_logic_vector(15 downto 0);
SIGNAL Buf_D2_Inc_PC : std_logic_vector(15 downto 0);
SIGNAL Buf_D_Link_PC : std_logic_vector(15 downto 0);
SIGNAL Link_PC : std_logic_vector(15 downto 0);
SIGNAL Buf_Link_PC : std_logic_vector(15 downto 0);


-- INSTANCES
BEGIN
In_PC <= Buf_In_PC;
PC <= Buf_PC;
D2_Inc_PC <= Buf_D2_Inc_PC;
D_Link_PC <= Buf_D_Link_PC;
Scan_Data_Out <= IAR(15);

halfword_reg_single_1 : word_reg_single   PORT MAP(
      Clock => Clock,
      Data_In => Buf_In_PC,
      Data_out => Buf_PC,
      Enable => Stalln,
      Resetn => Resetn,
      Scan_Data_In => Scan_Data_In,
      Scan_Enable => Scan_Enable
);
halfword_mux3_1 : word_mux3   PORT MAP(
      A => PC_Incr,
      B => ALU_Out,
      C => IAR,
      Out_word => Buf_In_PC,
      Sel => PC_Sel
);
halfword_increment_1 : increment   PORT MAP(
      CI => '1',
      In_word => Buf_PC,
      Out_word => PC_Incr
);

halfword_reg_single_2 : word_reg_single   PORT MAP(
      Clock => Clock,
      Data_In => PC_Incr,
      Data_out => Buf_D1_Inc_PC,
      Enable => Stalln,
      Resetn => Resetn,
      Scan_Data_In => Buf_PC(15),
      Scan_Enable => Scan_Enable
);
halfword_reg_single_3 : word_reg_single   PORT MAP(
      Clock => Clock,
      Data_In => Buf_D1_Inc_PC,
      Data_out => Buf_D2_Inc_PC,
      Enable => Stalln,
      Resetn => Resetn,
      Scan_Data_In => Buf_D1_Inc_PC(15),
      Scan_Enable => Scan_Enable
);
halfword_increment_2 : increment   PORT MAP(
      CI => '1',
      In_word(0) => '1',
      In_word(15 downto 1) => Buf_D2_Inc_PC(15 downto 1),
      Out_word(15 downto 0) => Link_PC(15 downto 0)
);
halfword_reg_single_4 : word_reg_single   PORT MAP(
      Clock => Clock,
      Data_In(0) => Buf_D2_Inc_PC(0),
      Data_In(15 downto 1)  => Link_PC(15 downto 1),
      Data_out => Buf_Link_PC,
      Enable => Stalln,
      Resetn => Resetn,
      Scan_Data_In => Buf_D2_Inc_PC(15),
      Scan_Enable => Scan_Enable
);
halfword_reg_single_5 : word_reg_single   PORT MAP(
      Clock => Clock,
      Data_In => Buf_Link_PC,
  Data_Out => Buf_D_Link_PC,
      Enable => Stalln,
      Resetn => Resetn,
      Scan_Data_In => Buf_Link_PC(15),
      Scan_Enable => Scan_Enable
);
halfword_reg_single_6 : word_reg_single   PORT MAP(
      Clock => Clock,
      Data_In => Buf_D_Link_PC,
      Data_out => IAR,
      Enable => IAR_Enable,
      Resetn => Resetn,
      Scan_Data_In => Buf_D_Link_PC(15),
      Scan_Enable => Scan_Enable
);
END structural;
```

12. pipeline

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

-- ***** pipeline model *****
-- external ports
ENTITY pipeline IS PORT (
      alu_op : OUT std_logic_vector(4 downto 0);
      A_Mux : OUT std_logic_vector(1 downto 0);
      B_Mux : OUT std_logic_vector(1 downto 0);

      Clock : IN std_logic;
      Data_In : IN std_logic_vector(23 downto 0);
      Dest : OUT std_logic_vector(3 downto 0);
      Immed : OUT std_logic_vector(15 downto 0);
      PC_Sel : OUT std_logic_vector(1 downto 0);
      rd_enable : OUT std_logic;
      Reg_In_Sel : OUT std_logic_vector(1 downto 0);
      Resetn : IN std_logic;
      RSone : OUT std_logic_vector(3 downto 0);
      RStwo : OUT std_logic_vector(3 downto 0);
      Scan_Data_In : IN std_logic;
      Scan_Enable : IN std_logic;
      Stalln : IN std_logic;
      wb_enable : OUT std_logic;
      scan_out : OUT std_logic;
      IAR_Enable : OUT std_logic;
      wr_enable : OUT std_logic;
      zero_flag : IN std_logic
);
END pipeline;

-- internal structure
ARCHITECTURE rtl OF pipeline IS

-- COMPONENTS

COMPONENT twelve_bit_reg_single
PORT (
      Clock : IN std_logic;
      Data_In : IN std_logic_vector(11 downto 0);
      Data_out : OUT std_logic_vector(11 downto 0);
      Enable : IN std_logic;
      Resetn : IN std_logic;
      Scan_Data_In : IN std_logic;
      Scan_Enable : IN std_logic
);
END COMPONENT;
```

```
	COMPONENT twenty_four_bit_reg_single
	PORT (
		Clock : IN std_logic;
		Data_In : IN std_logic_vector(23 downto 0);
		Data_out : OUT std_logic_vector(23 downto 0);
		Enable : IN std_logic;
		Resetn : IN std_logic;
		Scan_Data_In : IN std_logic;
		Scan_Enable : IN std_logic
	);
	END COMPONENT;


	-- SIGNALS
	SIGNAL Dec_Instr : std_logic_vector (23 downto 0);
	SIGNAL Ex_Instr : std_logic_vector (23 downto 0);
	SIGNAL Mem_Instr : std_logic_vector (11 downto 0);
	SIGNAL WB_Instr :   std_logic_vector (11 downto 0);



	-- INSTANCES
	BEGIN

	----- ****** decode pipeline stage *********

	twenty_bit_reg_single_1 : twenty_four_bit_reg_single   PORT MAP(
		Clock => Clock,
		Data_In => Data_In,
	  Data_out => Dec_Instr,
		Enable => Stalln,
		Resetn => Resetn,
		Scan_Data_In => Scan_Data_In,
		Scan_Enable => Scan_Enable
	);

	process (Dec_Instr)
	begin
	RSone <= Dec_Instr(15 downto 12);

	---- assign RS2 (check for SW instruction)
	if (Dec_Instr(23 downto 16) = X"45") then
	  RStwo <= Dec_Instr(11 downto 8) ;
	else RStwo <=  Dec_Instr(7 downto 4);
	end if;
	end process;
	------ ****** execute  pipeline stage *********

	twenty_four_bit_reg_single_2 : twenty_four_bit_reg_single   PORT
MAP(
		Clock => Clock,
		Data_In => Dec_Instr,
		Data_out => Ex_Instr,
		Enable => Stalln,
		Resetn => Resetn,
		Scan_Data_In => Dec_Instr(23),
		Scan_Enable => Scan_Enable
```

```
      );

      Immed <= Ex_Instr(15 downto 0);   ---- assign immediate value
      alu_op <= Ex_Instr(20 downto 16);  ---- assign alu opcodes
      b_mux <= Ex_Instr(22 downto 21);    --- assign b_mux


      PC_Sel <= "01" when Ex_Instr(23 downto 16) = X"C8" else  -----
when OP_J
                      "01" when Ex_Instr(23 downto 16) = X"E8" else ---
-- when OP_JAL
                      "0" & zero_flag when Ex_Instr(23 downto 16) =
X"C1" else ---when OP_BEQZ
                      "0" & not(zero_flag) when Ex_Instr(23 downto 16)
= X"C0" else ---when OP_BEQZ
                       "10" when Ex_Instr(23 downto 16) = X"F8" else --
-OP_RFE
                        "01" when Ex_Instr(23 downto 16) = X"28" else -
--- OP_TRAP
                        "01" when Ex_Instr(23 downto 16) = X"48" else -
--- OP_JR
                         "01" when Ex_Instr(23 downto 16) = X"68" else
----OP_JALR
                          "00";

      process (Ex_Instr)
      begin

      case Ex_Instr(23 downto 16) is
        when X"C8" =>           ----- when OP_J
           A_Mux  <= "11";
         when X"E8" =>           ----- when OP_JAL
           A_Mux  <= "11";
        when X"C1" =>           ----- when OP_BEQZ
           A_Mux  <= "01";
        when X"C0" =>           ----- when OP_BNEZ
           A_Mux  <= "01";
        when X"08" =>           ----- when OP_LHI
           A_Mux  <= "10";
        when X"F8" =>           ----- when OP_RFE
           A_Mux  <= "00";
        when X"28" =>           ----- when OP_TRAP
           A_Mux  <= "11";
        when X"48" =>           ----- when OP_JR
           A_Mux  <= "00";
        when X"68" =>           ----- when OP_JALR
           A_Mux  <= "00";
        when  others =>           ----- OTHERS
           A_Mux  <= "00";
      end case;
      end process;

      ------ ***** memory stage of pipeline ******* -----------

      twelve_bit_reg_single_1 : twelve_bit_reg_single   PORT MAP(
            Clock => Clock,
            Data_In(11 downto 4) => Ex_Instr(23 downto 16),
            Data_In(3 downto 0) => Ex_Instr(11 downto 8),
            Data_out => Mem_Instr,
            Enable => Stalln,
            Resetn => Resetn,
            Scan_Data_In => Ex_Instr(23),
            Scan_Enable => Scan_Enable
      );

      process (Mem_Instr)
      begin
      case Mem_Instr(11 downto 4) is
        when X"45" =>
           rd_enable <= '0';     ----- OP_SW (write)
           wr_enable <= '1';
         when X"44" =>           ------ OP_LW (read)
           rd_enable <= '1';
           wr_enable <= '0';
         when others =>
           rd_enable <= '0';
           wr_enable <= '0';
      end case;
      end process;

      ------ ******** write back stage ********
      twelve_bit_reg_single_2 : twelve_bit_reg_single   PORT MAP(
            Clock => Clock,
            Data_In => Mem_Instr,
            Data_out => WB_Instr,
            Enable => Stalln,
            Resetn => Resetn,
            Scan_Data_In => Mem_Instr(11),
            Scan_Enable => Scan_Enable
      );

      scan_out <= WB_Instr(11);
      process (WB_Instr)
      begin

      ---- check for Jump and Link Instructions to set Reg_In_Sel(0) =
0
        if (WB_Instr(11 downto 4) = X"E8" or WB_Instr(11 downto 4) =
X"68") then
           Reg_In_Sel(1) <= '1';
           Dest <= "1111";
        else Reg_In_Sel(1) <= '0';
           Dest <= WB_Instr(3 downto 0);
        end if;

      ---- check for TRAP to set IAR_Enable = 1
        if (WB_Instr(11 downto 4) = X"28") then
          IAR_Enable <= '1';
        else   IAR_Enable <= '0';
        end if;

      ---- check for LW to set Reg_In_Sel(1) = 1
        if (WB_Instr(11 downto 4) = X"44" ) then
           Reg_In_Sel(0) <= '1';
```

```
  else Reg_In_Sel(0) <= '0';
  end if;

------ set write_back enable
  case WB_Instr(11 downto 4) is
     when  X"C8" =>           ----- when OP_J
     WB_Enable <= '0';
  when X"C1" =>           ----- when OP_BEQZ
     WB_Enable <= '0';
  when X"C0" =>           ----- when OP_BNEZ
     WB_Enable <= '0';
  when X"45" =>           ----- when OP_SW
     WB_Enable <= '0';
  when X"F8" =>           ----- when OP_RFE
     WB_Enable <= '0';
  when X"28" =>           ----- when OP_TRAP
     WB_Enable <= '0';
  when X"48" =>           ----- when OP_JR
     WB_Enable <= '0';
  when X"00" =>           ----- when OP_NOP
     WB_Enable <= '0';
  when others =>
     WB_Enable <= '1';
  end case;
end process;
END rtl;
```

13. regfile

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

-----******* regfile model ***********

---- external ports
ENTITY regfile IS PORT (
  A : OUT std_logic_vector(15 downto 0);
  B : OUT std_logic_vector(15 downto 0);
  clock : IN std_logic;
  Data_In : IN std_logic_vector(15 downto 0);
  Dest : IN std_logic_vector(3 downto 0);
  stalln : IN std_logic;
  RSone : IN std_logic_vector(3 downto 0);
  RStwo : IN std_logic_vector(3 downto 0);
  scan_data_in : IN std_logic;
  scan_enable : IN std_logic;
  Resetn : IN std_logic;
  wb_enable : IN std_logic
);
END regfile;

---- internal structure
ARCHITECTURE structural OF regfile is

---- COMPONENTS
```

```
COMPONENT Dest_Decoder
PORT (
      Dest : IN std_logic_vector(3 downto 0);
      Enable : OUT std_logic_vector(15 downto 1);
      wb_enable : IN std_logic
);
END COMPONENT;

COMPONENT word_reg_single
PORT (
      Clock : IN std_logic;
      Data_In : IN std_logic_vector (15 downto 0);
      Data_out : OUT std_logic_vector (15 downto 0);
      enable : IN std_logic;
      Resetn : IN std_logic;
      Scan_Data_In : IN std_logic;
      Scan_Enable : IN std_logic
);
END COMPONENT;

COMPONENT word_mux16
PORT (
      In_Word0 : IN std_logic_vector(15 downto 0);
      In_Word1 : IN std_logic_vector(15 downto 0);
      In_Word2 : IN std_logic_vector(15 downto 0);
      In_Word3 : IN std_logic_vector(15 downto 0);
      In_Word4 : IN std_logic_vector(15 downto 0);
      In_Word5 : IN std_logic_vector(15 downto 0);
      In_Word6 : IN std_logic_vector(15 downto 0);
      In_Word7 : IN std_logic_vector(15 downto 0);
      In_Word8 : IN std_logic_vector(15 downto 0);
      In_Word9 : IN std_logic_vector(15 downto 0);
      In_Word10 : IN std_logic_vector(15 downto 0);
      In_Word11 : IN std_logic_vector(15 downto 0);
      In_Word12 : IN std_logic_vector(15 downto 0);
      In_Word13 : IN std_logic_vector(15 downto 0);
      In_Word14 : IN std_logic_vector(15 downto 0);
      In_Word15 : IN std_logic_vector(15 downto 0);
      Out_word : Out std_logic_vector(15 downto 0);
      Sel : IN std_logic_vector(3 downto 0)
);
END component;



----- signals
signal Enable : std_logic_vector(15 downto 1);
signal Reg1_Data : std_logic_vector(15 downto 0);
signal Reg2_Data : std_logic_vector(15 downto 0);
signal Reg3_Data : std_logic_vector(15 downto 0);
signal Reg4_Data : std_logic_vector(15 downto 0);
signal Reg5_Data : std_logic_vector(15 downto 0);
signal Reg6_Data : std_logic_vector(15 downto 0);
signal Reg7_Data : std_logic_vector(15 downto 0);
signal Reg8_Data : std_logic_vector(15 downto 0);
signal Reg9_Data : std_logic_vector(15 downto 0);
signal Reg10_Data : std_logic_vector(15 downto 0);
signal Reg11_Data : std_logic_vector(15 downto 0);
signal Reg12_Data : std_logic_vector(15 downto 0);
signal Reg13_Data : std_logic_vector(15 downto 0);
signal Reg14_Data : std_logic_vector(15 downto 0);
signal Reg15_Data : std_logic_vector(15 downto 0);
signal RegA_Data : std_logic_vector(15 downto 0);
signal MuxA_Data : std_logic_vector(15 downto 0);
signal MuxB_Data : std_logic_vector(15 downto 0);
signal zero_word :  std_logic_vector(15 downto 0);



begin

zero_word <= "0000000000000000";

---- port maps

Dest_Decoder1 :  Dest_Decoder PORT MAP (
      Dest=> Dest,
      Enable => Enable,
      wb_enable => wb_enable
);


word_reg1 : word_reg_single PORT MAP (
      Clock => clock,
      Data_In => Data_In,

      Data_out => Reg1_Data,
      Enable => Enable(1),
      Resetn => Resetn,
      Scan_Data_In => Scan_Data_In,
      Scan_Enable => Scan_Enable
);

word_reg2 : word_reg_single PORT MAP (
      Clock => clock,
      Data_In => Data_In,
      Data_out => Reg2_Data,
      Enable => Enable(2),
      Resetn => Resetn,
      Scan_Data_In => Reg1_Data(15),
      Scan_Enable => Scan_Enable
);

word_reg3 : word_reg_single PORT MAP (
      Clock => clock,
      Data_In => Data_In,
      Data_out => Reg3_Data,
      Enable => Enable(3),
      Resetn => Resetn,
      Scan_Data_In => Reg2_Data(15),
      Scan_Enable => Scan_Enable
);

word_reg4 : word_reg_single PORT MAP (
      Clock => clock,
      Data_In => Data_In,
      Data_out => Reg4_Data,
      Enable => Enable(4),
      Resetn => Resetn,
      Scan_Data_In => Reg3_Data(15),
      Scan_Enable => Scan_Enable
);

word_reg5 : word_reg_single PORT MAP (
      Clock => clock,
      Data_In => Data_In,
      Data_out => Reg5_Data,
      Enable => Enable(5),
      Resetn => Resetn,
      Scan_Data_In => Reg4_Data(15),
      Scan_Enable => Scan_Enable
);

word_reg6 : word_reg_single PORT MAP (
      Clock => clock,
      Data_In => Data_In,
      Data_out => Reg6_Data,
      Enable => Enable(6),
      Resetn => Resetn,
      Scan_Data_In => Reg5_Data(15),
      Scan_Enable => Scan_Enable
);

word_reg7 : word_reg_single PORT MAP (
      Clock => clock,
      Data_In => Data_In,
      Data_out => Reg7_Data,
      Enable => Enable(7),
      Resetn => Resetn,
      Scan_Data_In => Reg6_Data(15),
      Scan_Enable => Scan_Enable
);

word_reg8 : word_reg_single PORT MAP (
      Clock => clock,
      Data_In => Data_In,
      Data_out => Reg8_Data,
      Enable => Enable(8),
      Resetn => Resetn,
      Scan_Data_In => Reg7_Data(15),
      Scan_Enable => Scan_Enable
);

word_reg9 : word_reg_single PORT MAP (
      Clock => clock,
      Data_In => Data_In,
      Data_out => Reg9_Data,
      Enable => Enable(9),
      Resetn => Resetn,
      Scan_Data_In => Reg8_Data(15),
      Scan_Enable => Scan_Enable
```

```
);

word_reg10 : word_reg_single PORT MAP (
      Clock => clock,
      Data_In => Data_In,
      Data_out => Reg10_Data,
      Enable => Enable(10),
      Resetn => Resetn,
      Scan_Data_In => Reg9_Data(15),
      Scan_Enable => Scan_Enable
);

word_reg11 : word_reg_single PORT MAP (
      Clock => clock,
      Data_In => Data_In,
      Data_out => Reg11_Data,
      Enable => Enable(11),
      Resetn => Resetn,
      Scan_Data_In => Reg10_Data(15),
      Scan_Enable => Scan_Enable
);

word_reg12 : word_reg_single PORT MAP (
      Clock => clock,
      Data_In => Data_In,
      Data_out => Reg12_Data,
      Enable => Enable(12),
      Resetn => Resetn,
      Scan_Data_In => Reg11_Data(15),
      Scan_Enable => Scan_Enable
);

word_reg13 : word_reg_single PORT MAP (
      Clock => clock,
      Data_In => Data_In,
      Data_out => Reg13_Data,
      Enable => Enable(13),
      Resetn => Resetn,
      Scan_Data_In => Reg12_Data(15),
      Scan_Enable => Scan_Enable
);

word_reg14 : word_reg_single PORT MAP (
      Clock => clock,
      Data_In => Data_In,
      Data_out => Reg14_Data,
      Enable => Enable(14),
      Resetn => Resetn,
      Scan_Data_In => Reg13_Data(15),
      Scan_Enable => Scan_Enable
);

word_reg15 : word_reg_single PORT MAP (
      Clock => clock,
      Data_In => Data_In,
      Data_out => Reg15_Data,
      Enable => Enable(15),
      Resetn => Resetn,
      Scan_Data_In => Reg14_Data(15),
      Scan_Enable => Scan_Enable
);

word_regA : word_reg_single PORT MAP (
      Clock => clock,
      Data_In => MuxA_Data,
      Data_out => RegA_Data,
      Enable => stalln,
      Resetn => Resetn,
      Scan_Data_In => Reg15_Data(15),
      Scan_Enable => Scan_Enable
);

A <= RegA_Data;

word_regB : word_reg_single PORT MAP (
      Clock => clock,
      Data_In => MuxB_Data,
      Data_out => B,
      Enable =>  stalln,
      Resetn => Resetn,
      Scan_Data_In => RegA_Data(15),
      Scan_Enable => Scan_Enable
);

MuxA : word_mux16 PORT MAP (
  In_Word0    => zero_word,
      In_Word1    =>  Reg1_Data,
      In_Word2    =>  Reg2_Data,
      In_Word3    =>  Reg3_Data,
      In_Word4    =>  Reg4_Data,
      In_Word5    =>  Reg5_Data,
      In_Word6    =>  Reg6_Data,
      In_Word7    =>  Reg7_Data,
      In_Word8    =>  Reg8_Data,
      In_Word9    =>  Reg9_Data,
      In_Word10  =>  Reg10_Data,
      In_Word11  =>  Reg11_Data,
      In_Word12  =>  Reg12_Data,
      In_Word13  =>  Reg13_Data,
      In_Word14  =>  Reg14_Data,
      In_Word15  =>  Reg15_Data,
      Out_word    =>  MuxA_Data,
      Sel => RSone
);

MuxB : word_mux16 PORT MAP (
  In_Word0    =>  zero_word,
      In_Word1    =>  Reg1_Data,
      In_Word2    =>  Reg2_Data,
      In_Word3    =>  Reg3_Data,
      In_Word4    =>  Reg4_Data,
      In_Word5    =>  Reg5_Data,
      In_Word6    =>  Reg6_Data,
      In_Word7    =>  Reg7_Data,
      In_Word8    =>  Reg8_Data,
      In_Word9    =>  Reg9_Data,
      In_Word10  =>  Reg10_Data,
      In_Word11  =>  Reg11_Data,
      In_Word12  =>  Reg12_Data,
      In_Word13  =>  Reg13_Data,
      In_Word14  =>  Reg14_Data,
      In_Word15  =>  Reg15_Data,
      Out_word    =>  MuxB_Data,
      Sel => RStwo
);

END structural;
```

14. rwcontrol

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

-- ***** rw_control model *****
-- external ports
ENTITY rw_control IS PORT (
      Clock : IN std_logic;
      Prog_Rd : OUT std_logic;
      Rd : OUT std_logic;
      rd_enable : IN std_logic;
      resetn : IN std_logic;
      stalln : IN std_logic;
      Wr : OUT std_logic;
      wr_enable : IN std_logic
);
END rw_control;

-- internal structure
ARCHITECTURE rtl OF rw_control IS



-- SIGNALS

SIGNAL clockn : std_logic; --- inverted clock

BEGIN
clockn <= not(Clock);
Wr <=  not (clockn and wr_enable);
Rd <=  not (clockn and rd_enable);
Prog_Rd <= not (clockn and resetn and stalln);
end rtl;
```

15. scan_reg

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

-- ***** scan_reg model *****
-- external ports
```

```
ENTITY scan_reg IS PORT (
      clk : IN std_logic;
      data_in : IN std_logic;
      data_out : OUT std_logic;
      enable : IN std_logic;
      resetn : IN std_logic;
      scan_data_in : IN std_logic;
      scan_enable : IN std_logic
);
END scan_reg;

-- internal structure
ARCHITECTURE rtl OF scan_reg IS


-- INSTANCES
BEGIN

process (clk, resetn)
begin
if (resetn = '0') then
  data_out <= '0';
elsif (clk = '1' and clk'event) then
  if (scan_enable = '1') then
    data_out <= scan_data_in;
  elsif (enable = '1') then
     data_out <= data_in;
  end if;
 end if;
end process;

END rtl;
```

16. twelve_bit_reg_single

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

-- ***** twelve_bit_reg_single model *****
-- external ports
ENTITY twelve_bit_reg_single IS PORT (
      Clock : IN std_logic;
      Data_In : IN std_logic_vector(11 downto 0);
      Data_out : OUT std_logic_vector(11 downto 0);
      Enable : IN std_logic;
      Resetn : IN std_logic;
      Scan_Data_In : IN std_logic;
      Scan_Enable : IN std_logic
);
END twelve_bit_reg_single;

-- internal structure
ARCHITECTURE structural OF twelve_bit_reg_single IS

-- COMPONENTS
```

```
COMPONENT scan_reg
PORT (
      clk : IN std_logic;
      data_in : IN std_logic;
      data_out : OUT std_logic;
      enable : IN std_logic;
      resetn : IN std_logic;
      scan_data_in : IN std_logic;
      scan_enable : IN std_logic
);
END COMPONENT;

-- SIGNALS


signal buf_data_out : std_logic_vector (10 downto 0);




-- INSTANCES
BEGIN
Data_out(0) <= buf_data_out(0);
Data_out(1) <= buf_data_out(1);
Data_out(2) <= buf_data_out(2);
Data_out(3) <= buf_data_out(3);
Data_out(4) <= buf_data_out(4);
Data_out(5) <= buf_data_out(5);
Data_out(6) <= buf_data_out(6);
Data_out(7) <= buf_data_out(7);
Data_out(8) <= buf_data_out(8);
Data_out(9) <= buf_data_out(9);
Data_out(10) <= buf_data_out(10);



scan_reg_1 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(1),
      data_out => buf_data_out(1),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => buf_data_out(0),
      scan_enable => Scan_Enable
);
scan_reg_2 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(2),
      data_out => buf_data_out(2),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => buf_data_out(1),
      scan_enable => Scan_Enable
);
scan_reg_3 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(3),
      data_out => buf_data_out(3),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => buf_data_out(2),
      scan_enable => Scan_Enable
);
scan_reg_4 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(4),
      data_out => buf_data_out(4),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => buf_data_out(3),
      scan_enable => Scan_Enable
);
scan_reg_5 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(0),
      data_out => buf_data_out(0),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => Scan_Data_In,
      scan_enable => Scan_Enable
);
scan_reg_6 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(5),
      data_out => buf_data_out(5),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => buf_data_out(4),
      scan_enable => Scan_Enable
);
scan_reg_7 : scan_reg   PORT MAP(
      clk => Clock,

      data_in => Data_In(6),
      data_out => buf_data_out(6),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => buf_data_out(5),
      scan_enable => Scan_Enable
);
scan_reg_8 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(7),
      data_out => buf_data_out(7),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => buf_data_out(6),
      scan_enable => Scan_Enable
);
scan_reg_9 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(8),
      data_out => buf_data_out(8),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => buf_data_out(7),
      scan_enable => Scan_Enable
);
scan_reg_10 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(9),
      data_out => buf_data_out(9),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => buf_data_out(8),
      scan_enable => Scan_Enable
);
scan_reg_11 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(10),
      data_out => buf_data_out(10),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => buf_data_out(9),
      scan_enable => Scan_Enable

);
scan_reg_12 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(11),
      data_out => Data_out(11),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => buf_data_out(10),
      scan_enable => Scan_Enable
);
END structural;
```

17. twenty_four_bit_reg_single

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

-- ***** twenty_four_bit_reg_single model *****
-- external ports
ENTITY twenty_four_bit_reg_single IS PORT (
      Clock : IN std_logic;
      Data_In : IN std_logic_vector (23 downto 0);
  Data_out : OUT std_logic_vector (23 downto 0);
      Enable : IN std_logic;
      Resetn : IN std_logic;
      Scan_Data_In : IN std_logic;
      Scan_Enable : IN std_logic
);
END twenty_four_bit_reg_single;

-- internal structure
ARCHITECTURE structural OF twenty_four_bit_reg_single IS
```

```
-- COMPONENTS

Component twelve_bit_reg_single
 PORT (
      Clock : IN std_logic;
      Data_In : IN std_logic_vector(11 downto 0);
      Data_out : OUT std_logic_vector(11 downto 0);
      Enable : IN std_logic;
      Resetn : IN std_logic;
      Scan_Data_In : IN std_logic;
      Scan_Enable : IN std_logic
);
END Component;



-- SIGNALS
SIGNAL Buf_Data_out11 : std_logic;


-- INSTANCES
BEGIN
Data_out(11) <= Buf_Data_out11;

twelve_bit_reg_single1 : twelve_bit_reg_single PORT MAP(
  Clock => Clock,
  Data_In => Data_In(11 downto 0),
  Data_Out(10 downto 0) => Data_Out(10 downto 0),
  Data_Out(11) => Buf_Data_out11,
  Enable => Enable,
  Resetn => Resetn,
  Scan_Data_In => Scan_Data_In,
  Scan_Enable => Scan_Enable
);

twelve_bit_reg_single2 : twelve_bit_reg_single PORT MAP(
  Clock => Clock,
  Data_In => Data_In(23 downto 12),
  Data_Out => Data_Out(23 downto 12),
  Enable => Enable,
  Resetn => Resetn,
  Scan_Data_In => Buf_Data_out11,
  Scan_Enable => Scan_Enable
);
END structural;
```

18. word_mux16

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

-- ***** word_mux16 model *****
-- external ports
ENTITY word_mux16 IS PORT (
      In_Word0 : IN std_logic_vector(15 downto 0);
      In_Word1 : IN std_logic_vector(15 downto 0);
```

```
        In_Word2 : IN std_logic_vector(15 downto 0);
        In_Word3 : IN std_logic_vector(15 downto 0);
        In_Word4 : IN std_logic_vector(15 downto 0);
        In_Word5 : IN std_logic_vector(15 downto 0);
        In_Word6 : IN std_logic_vector(15 downto 0);
        In_Word7 : IN std_logic_vector(15 downto 0);
        In_Word8 : IN std_logic_vector(15 downto 0);
        In_Word9 : IN std_logic_vector(15 downto 0);
        In_Word10 : IN std_logic_vector(15 downto 0);
        In_Word11 : IN std_logic_vector(15 downto 0);
        In_Word12 : IN std_logic_vector(15 downto 0);
        In_Word13 : IN std_logic_vector(15 downto 0);
        In_Word14 : IN std_logic_vector(15 downto 0);
        In_Word15 : IN std_logic_vector(15 downto 0);
        Out_word : Out std_logic_vector(15 downto 0);
        Sel : IN std_logic_vector(3 downto 0)
);
END word_mux16;

-- internal structure
ARCHITECTURE rtl OF word_mux16 IS
BEGIN

with sel select
   Out_word <= In_Word0 when  "0000",
                       In_Word1 when  "0001",
                                                In_Word2 when
"0010",
                       In_Word3 when  "0011",
                       In_Word4 when  "0100",
                       In_Word5 when  "0101",
                       In_Word6 when  "0110",
                       In_Word7 when  "0111",
                       In_Word8 when  "1000",
                       In_Word9 when  "1001",

                       In_Word10 when "1010",
                       In_Word11 when "1011",
                       In_Word12 when "1100",
                       In_Word13 when "1101",
                       In_Word14 when "1110",
                       In_Word15 when others;

END rtl;
```

19. word_mux3

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

-- ***** word_mux3 model *****
-- external ports
ENTITY word_mux3 IS PORT (
      A : IN std_logic_vector(15 downto 0);
      B : IN std_logic_vector(15 downto 0);
      C : IN std_logic_vector(15 downto 0);
      Out_word : Out std_logic_vector(15 downto 0);
      Sel : IN std_logic_vector(1 downto 0)
);
END word_mux3;

-- internal structure
ARCHITECTURE rtl OF word_mux3 IS
BEGIN
process (A, B, C, Sel)
begin
case sel is
  when "00" => Out_word <= A;
  when "01" => Out_word <= B;
  when others => Out_word <= C;
end case;
end process;
END rtl;
```

20. word_mux4.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

-- ***** word_mux4 model *****
-- external ports
ENTITY word_mux4 IS PORT (
      A : IN std_logic_vector(15 downto 0);
      B : IN std_logic_vector(15 downto 0);
      C : IN std_logic_vector(15 downto 0);
      D : IN std_logic_vector(15 downto 0);
      Out_word : Out std_logic_vector(15 downto 0);
      Sel : IN std_logic_vector(1 downto 0)
);
END word_mux4;

-- internal structure
ARCHITECTURE rtl OF word_mux4 IS
BEGIN
process (A, B, C, D, Sel)
begin
case sel is
  when "00" => Out_word <= A;
  when "01" => Out_word <= B;
  when "10" => Out_word <= C;
  when others => Out_word <= D;
end case;
end process;
END rtl;
```

21. word_reg_single.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

-- ***** word_reg_single model *****
-- external ports
ENTITY word_reg_single IS PORT (
      Clock : IN std_logic;
      Data_In : IN std_logic_vector (15 downto 0);
      Data_out : OUT std_logic_vector (15 downto 0);
      Enable : IN std_logic;
      Resetn : IN std_logic;
      Scan_Data_In : IN std_logic;
      Scan_Enable : IN std_logic
);
END word_reg_single;

-- internal structure
ARCHITECTURE structural OF word_reg_single IS

-- COMPONENTS

COMPONENT scan_reg
PORT (
      clk : IN std_logic;
      data_in : IN std_logic;
      data_out : OUT std_logic;
      enable : IN std_logic;
      resetn : IN std_logic;
      scan_data_in : IN std_logic;
      scan_enable : IN std_logic
);
END COMPONENT;



-- SIGNALS

SIGNAL Buf_Data_out : std_logic_vector(14 downto 0);




-- INSTANCES
BEGIN

Data_out(0) <= Buf_Data_out(0);
Data_out(1) <= Buf_Data_out(1);
Data_out(2) <= Buf_Data_out(2);
Data_out(3) <= Buf_Data_out(3);
Data_out(4) <= Buf_Data_out(4);
Data_out(5) <= Buf_Data_out(5);
Data_out(6) <= Buf_Data_out(6);
Data_out(7) <= Buf_Data_out(7);
Data_out(8) <= Buf_Data_out(8);
Data_out(9) <= Buf_Data_out(9);
Data_out(10) <= Buf_Data_out(10);
Data_out(11) <= Buf_Data_out(11);
Data_out(12) <= Buf_Data_out(12);
Data_out(13) <= Buf_Data_out(13);
Data_out(14) <= Buf_Data_out(14);
```

```
scan_reg_1 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(1),
      data_out => Buf_Data_out(1),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => Buf_Data_out(0),
      scan_enable => Scan_Enable
);
scan_reg_2 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(2),
      data_out => Buf_Data_out(2),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => Buf_Data_out(1),
      scan_enable => Scan_Enable
);
scan_reg_3 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(3),
      data_out => Buf_Data_out(3),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => Buf_Data_out(2),
      scan_enable => Scan_Enable
);
scan_reg_4 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(4),
      data_out => Buf_Data_out(4),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => Buf_Data_out(3),
      scan_enable => Scan_Enable
);
scan_reg_6 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(5),
      data_out => Buf_Data_out(5),

      enable => Enable,
      resetn => Resetn,
      scan_data_in => Buf_Data_out(4),
      scan_enable => Scan_Enable
);
scan_reg_7 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(6),
      data_out => Buf_Data_out(6),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => Buf_Data_out(5),
      scan_enable => Scan_Enable
);
scan_reg_8 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(7),
      data_out => Buf_Data_out(7),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => Buf_Data_out(6),
      scan_enable => Scan_Enable
);
scan_reg_9 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(8),
      data_out => Buf_Data_out(8),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => Buf_Data_out(7),
      scan_enable => Scan_Enable
);
scan_reg_10 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(9),
      data_out => Buf_Data_out(9),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => Buf_Data_out(8),
      scan_enable => Scan_Enable
);
scan_reg_11 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(10),
      data_out => Buf_Data_out(10),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => Buf_Data_out(9),
      scan_enable => Scan_Enable
);
scan_reg_12 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(11),
      data_out => Buf_Data_out(11),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => Buf_Data_out(10),
      scan_enable => Scan_Enable
);
scan_reg_13 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(12),
      data_out => Buf_Data_out(12),
      enable => Enable,

      resetn => Resetn,
      scan_data_in => Buf_Data_out(11),
      scan_enable => Scan_Enable
);
scan_reg_14 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(13),
      data_out => Buf_Data_out(13),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => Buf_Data_out(12),
      scan_enable => Scan_Enable
);
scan_reg_15 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(14),
      data_out => Buf_Data_out(14),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => Buf_Data_out(13),
      scan_enable => Scan_Enable
);
scan_reg_16 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(15),
      data_out => Data_out(15),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => Buf_Data_out(14),
      scan_enable => Scan_Enable
);
scan_reg_5 : scan_reg   PORT MAP(
      clk => Clock,
      data_in => Data_In(0),
      data_out => Buf_Data_out(0),
      enable => Enable,
      resetn => Resetn,
      scan_data_in => Scan_Data_In,
      scan_enable => Scan_Enable
);
END structural;
```

22. word_set.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

-- ***** word_set model *****
-- external ports
ENTITY word_set IS PORT (
      In_word : IN std_logic_vector (15 downto 0);
      set_op : IN std_logic_vector (2 downto 0);
      set_out : OUT std_logic
);
END word_set;

-- internal structure
ARCHITECTURE rtl OF word_set IS

component zero_test
PORT (
      In_word : in std_logic_vector(15 downto 0);
      zero_flag : OUT std_logic
);
```

```
END component;
signal zero_flag : std_logic;

begin
process (In_word, set_op, zero_flag)
begin
case set_op is
  when "000" => set_out <= zero_flag;
  when "001" => set_out <= (not(In_word(15)) or zero_flag);
  when "010" => set_out <= not(In_word(15)) and not(zero_flag);
  when "011" => set_out <= (In_word(15) or zero_flag);

  when "100" => set_out <= In_word(15);
  when others => set_out <= not(zero_flag);
end case;
end process;
zero_test1 : zero_test port map (
                  In_word => In_word,
                   zero_flag => zero_flag
);



END rtl;
```

23. zero_test.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

-- ***** zero_test model *****
-- external ports
ENTITY zero_test IS PORT (
      In_word : in std_logic_vector(15 downto 0);
      zero_flag : OUT std_logic
);
END zero_test;

-- internal structure
ARCHITECTURE rtl OF zero_test IS
begin

process (In_word)
begin
  if (In_word = "0000000000000000") then
    zero_flag <= '1';
  else zero_flag <= '0';
  end if;
end process;

END rtl;
```

24. interrup.vhd

```
--  C:\XILINX\FULL_THING\INTERRUP.vhd
--  VHDL code created by Xilinx's StateCAD 5.1i
--  Thu Feb 27 09:36:14 2003

--  This VHDL code (for use with Xilinx XST) was generated using:
--  enumerated state assignment with structured code format.
--  Minimization is enabled,  implied else is enabled,
--  and outputs are speed optimized.

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY SHELL_INTERRUP IS
      PORT (CLK,E,RESET,RFE: IN std_logic;

DataAccess,ErrSyn1,ErrSyn2,ErrSyn3,ErrSyn4,Instr0,Instr1,Instr2,I
nstr3,

Instr4,Instr5,Instr6,Instr7,Instr8,Instr9,Instr10,Instr11,Instr12
,Instr13,

Instr14,Instr15,Instr16,Instr17,Instr18,Instr19,Instr20,Instr21,I
nstr22,
                  Instr23,InstrAccess,TRAP : OUT std_logic);
END;

ARCHITECTURE BEHAVIOR OF SHELL_INTERRUP IS
      TYPE type_sreg IS
(ErrSyndSave1,ErrSyndSave2,ErrSyndSave3,ErrSyndSave4,

InterrInstr,ISRData,ISRErrSyndSave1,ISRErrSyndSave2,ISRErrSyndSav
e3,
            ISRErrSyndSave4,ISRInst,NormData,NormInst);
      SIGNAL sreg, next_sreg : type_sreg;
      SIGNAL
next_BP_DataAccess,next_BP_ErrSyn1,next_BP_ErrSyn2,next_BP_ErrSyn3,

next_BP_ErrSyn4,next_BP_Instr0,next_BP_Instr1,next_BP_Instr2,next
_BP_Instr3,

next_BP_Instr4,next_BP_Instr5,next_BP_Instr6,next_BP_Instr7,next_
BP_Instr8,

next_BP_Instr9,next_BP_Instr10,next_BP_Instr11,next_BP_Instr12,

next_BP_Instr13,next_BP_Instr14,next_BP_Instr15,next_BP_Instr16,

next_BP_Instr17,next_BP_Instr18,next_BP_Instr19,next_BP_Instr20,

next_BP_Instr21,next_BP_Instr22,next_BP_Instr23,next_BP_InstrAcce
ss,
            next_BP_TRAP : std_logic;
      SIGNAL BP_Instr : std_logic_vector (23 DOWNTO 0);
      SIGNAL Instr : std_logic_vector (23 DOWNTO 0);
```

```
            SIGNAL
BP_DataAccess,BP_ErrSyn1,BP_ErrSyn2,BP_ErrSyn3,BP_ErrSyn4,BP_Instr0,

      BP_Instr1,BP_Instr2,BP_Instr3,BP_Instr4,BP_Instr5,BP_Instr6,BP_In
str7,

      BP_Instr8,BP_Instr9,BP_Instr10,BP_Instr11,BP_Instr12,BP_Instr13,B
P_Instr14,

      BP_Instr15,BP_Instr16,BP_Instr17,BP_Instr18,BP_Instr19,BP_Instr20
,BP_Instr21,
                  BP_Instr22,BP_Instr23,BP_InstrAccess,BP_TRAP:
std_logic;
      BEGIN
            PROCESS (CLK, RESET, next_sreg, next_BP_DataAccess,
next_BP_ErrSyn1,
                  next_BP_ErrSyn2, next_BP_ErrSyn3, next_BP_ErrSyn4,
next_BP_InstrAccess,
                  next_BP_TRAP, next_BP_Instr23, next_BP_Instr22,
next_BP_Instr21,
                  next_BP_Instr20, next_BP_Instr19, next_BP_Instr18,
next_BP_Instr17,
                  next_BP_Instr16, next_BP_Instr15, next_BP_Instr14,
next_BP_Instr13,
                  next_BP_Instr12, next_BP_Instr11, next_BP_Instr10,
next_BP_Instr9,
                  next_BP_Instr8, next_BP_Instr7, next_BP_Instr6,
next_BP_Instr5,
                  next_BP_Instr4, next_BP_Instr3, next_BP_Instr2,
next_BP_Instr1,
                  next_BP_Instr0)
            BEGIN
                  IF ( RESET='1' ) THEN
                        sreg <= NormInst;
                        BP_DataAccess <= '0';
                        BP_ErrSyn1 <= '0';
                        BP_ErrSyn2 <= '0';
                        BP_ErrSyn3 <= '0';
                        BP_ErrSyn4 <= '0';
                        BP_TRAP <= '0';
                        BP_Instr23 <= '0';
                        BP_Instr22 <= '0';
                        BP_Instr21 <= '0';
                        BP_Instr20 <= '0';
                        BP_Instr19 <= '0';
                        BP_Instr18 <= '0';
                        BP_Instr17 <= '0';
                        BP_Instr16 <= '0';
                        BP_Instr15 <= '0';
                        BP_Instr14 <= '0';
                        BP_Instr13 <= '0';
                        BP_Instr12 <= '0';
                        BP_Instr11 <= '0';
                        BP_Instr10 <= '0';
                        BP_Instr9 <= '0';
                        BP_Instr8 <= '0';
                        BP_Instr7 <= '0';
                        BP_Instr6 <= '0';
                        BP_Instr5 <= '0';
                        BP_Instr4 <= '0';
                        BP_Instr3 <= '0';
                        BP_Instr2 <= '0';
                        BP_Instr1 <= '0';
                        BP_Instr0 <= '0';
                        BP_InstrAccess <= '1';
                  ELSIF CLK='1' AND CLK'event THEN
                        sreg <= next_sreg;
                        BP_DataAccess <= next_BP_DataAccess;
                        BP_ErrSyn1 <= next_BP_ErrSyn1;
                        BP_ErrSyn2 <= next_BP_ErrSyn2;
                        BP_ErrSyn3 <= next_BP_ErrSyn3;
                        BP_ErrSyn4 <= next_BP_ErrSyn4;
                        BP_InstrAccess <= next_BP_InstrAccess;
                        BP_TRAP <= next_BP_TRAP;
                        BP_Instr23 <= next_BP_Instr23;
                        BP_Instr22 <= next_BP_Instr22;
                        BP_Instr21 <= next_BP_Instr21;
                        BP_Instr20 <= next_BP_Instr20;
                        BP_Instr19 <= next_BP_Instr19;
                        BP_Instr18 <= next_BP_Instr18;
                        BP_Instr17 <= next_BP_Instr17;
                        BP_Instr16 <= next_BP_Instr16;
                        BP_Instr15 <= next_BP_Instr15;
                        BP_Instr14 <= next_BP_Instr14;
                        BP_Instr13 <= next_BP_Instr13;
                        BP_Instr12 <= next_BP_Instr12;
                        BP_Instr11 <= next_BP_Instr11;
                        BP_Instr10 <= next_BP_Instr10;
                        BP_Instr9 <= next_BP_Instr9;
                        BP_Instr8 <= next_BP_Instr8;
                        BP_Instr7 <= next_BP_Instr7;
                        BP_Instr6 <= next_BP_Instr6;
                        BP_Instr5 <= next_BP_Instr5;
                        BP_Instr4 <= next_BP_Instr4;
                        BP_Instr3 <= next_BP_Instr3;
                        BP_Instr2 <= next_BP_Instr2;
                        BP_Instr1 <= next_BP_Instr1;
                        BP_Instr0 <= next_BP_Instr0;
                  END IF;
            END PROCESS;

            PROCESS
(sreg,BP_DataAccess,BP_ErrSyn1,BP_ErrSyn2,BP_ErrSyn3,BP_ErrSyn4,

      BP_Instr0,BP_Instr1,BP_Instr2,BP_Instr3,BP_Instr4,BP_Instr5,BP_In
str6,

      BP_Instr7,BP_Instr8,BP_Instr9,BP_Instr10,BP_Instr11,BP_Instr12,BP
_Instr13,

      BP_Instr14,BP_Instr15,BP_Instr16,BP_Instr17,BP_Instr18,BP_Instr19
,BP_Instr20,
      BP_Instr21,BP_Instr22,BP_Instr23,BP_InstrAccess,BP_TRAP,E,RFE,BP_
Instr)
            BEGIN
                  next_BP_DataAccess <= BP_DataAccess;next_BP_ErrSyn1
<= BP_ErrSyn1;
                        next_BP_ErrSyn2 <= BP_ErrSyn2;next_BP_ErrSyn3
<= BP_ErrSyn3;next_BP_ErrSyn4
                        <= BP_ErrSyn4;next_BP_Instr0 <=
BP_Instr0;next_BP_Instr1 <= BP_Instr1;
                        next_BP_Instr2 <= BP_Instr2;next_BP_Instr3 <=
BP_Instr3;next_BP_Instr4 <=
                        BP_Instr4;next_BP_Instr5 <=
BP_Instr5;next_BP_Instr6 <= BP_Instr6;
                        next_BP_Instr7 <= BP_Instr7;next_BP_Instr8 <=
BP_Instr8;next_BP_Instr9 <=
                        BP_Instr9;next_BP_Instr10 <=
BP_Instr10;next_BP_Instr11 <= BP_Instr11;
                        next_BP_Instr12 <= BP_Instr12;next_BP_Instr13
<= BP_Instr13;next_BP_Instr14
                        <= BP_Instr14;next_BP_Instr15 <=
BP_Instr15;next_BP_Instr16 <= BP_Instr16;
                        next_BP_Instr17 <= BP_Instr17;next_BP_Instr18
<= BP_Instr18;next_BP_Instr19
                        <= BP_Instr19;next_BP_Instr20 <=
BP_Instr20;next_BP_Instr21 <= BP_Instr21;
                        next_BP_Instr22 <= BP_Instr22;next_BP_Instr23
<= BP_Instr23;
                        next_BP_InstrAccess <=
BP_InstrAccess;next_BP_TRAP <= BP_TRAP;

                  BP_Instr <= (( std_logic_vector'(BP_Instr23,
BP_Instr22, BP_Instr21,
                        BP_Instr20, BP_Instr19, BP_Instr18, BP_Instr17,
BP_Instr16, BP_Instr15,
                        BP_Instr14, BP_Instr13, BP_Instr12, BP_Instr11,
BP_Instr10, BP_Instr9,
                        BP_Instr8, BP_Instr7, BP_Instr6, BP_Instr5,
BP_Instr4, BP_Instr3, BP_Instr2,
                        BP_Instr1, BP_Instr0)));


                  next_sreg<=ErrSyndSave1;

                  CASE sreg IS
                        WHEN ErrSyndSave1 =>
                              next_sreg<=ErrSyndSave2;
                              next_BP_ErrSyn1<='0';
                              next_BP_ErrSyn2<='1';

                              IF (( BP_TRAP='1' )) THEN
next_BP_TRAP<='1';
                              ELSE next_BP_TRAP<='0';
                              END IF;

                              IF (( BP_InstrAccess='1' )) THEN
next_BP_InstrAccess<='1';
```

```
                              ELSE next_BP_InstrAccess<='0';
                              END IF;

                              BP_Instr <= ((
std_logic_vector'(BP_Instr23, BP_Instr22, BP_Instr21,
                                    BP_Instr20, BP_Instr19, BP_Instr18,
BP_Instr17, BP_Instr16, BP_Instr15,
                                    BP_Instr14, BP_Instr13, BP_Instr12,
BP_Instr11, BP_Instr10, BP_Instr9,
                                    BP_Instr8, BP_Instr7, BP_Instr6,
BP_Instr5, BP_Instr4, BP_Instr3, BP_Instr2,
                                    BP_Instr1, BP_Instr0)));
                              IF (( BP_ErrSyn4='1' )) THEN
next_BP_ErrSyn4<='1';
                              ELSE next_BP_ErrSyn4<='0';
                              END IF;

                              IF (( BP_ErrSyn3='1' )) THEN
next_BP_ErrSyn3<='1';
                              ELSE next_BP_ErrSyn3<='0';
                              END IF;

                              IF (( BP_DataAccess='1' )) THEN
next_BP_DataAccess<='1';
                              ELSE next_BP_DataAccess<='0';
                              END IF;

                        WHEN ErrSyndSave2 =>
                              next_sreg<=ErrSyndSave3;
                              next_BP_ErrSyn2<='0';
                              next_BP_ErrSyn3<='1';

                              IF (( BP_TRAP='1' )) THEN
next_BP_TRAP<='1';
                              ELSE next_BP_TRAP<='0';
                              END IF;

                              IF (( BP_InstrAccess='1' )) THEN
next_BP_InstrAccess<='1';
                              ELSE next_BP_InstrAccess<='0';
                              END IF;

                              BP_Instr <= ((
std_logic_vector'(BP_Instr23, BP_Instr22, BP_Instr21,
                                    BP_Instr20, BP_Instr19, BP_Instr18,
BP_Instr17, BP_Instr16, BP_Instr15,
                                    BP_Instr14, BP_Instr13, BP_Instr12,
BP_Instr11, BP_Instr10, BP_Instr9,
                                    BP_Instr8, BP_Instr7, BP_Instr6,
BP_Instr5, BP_Instr4, BP_Instr3, BP_Instr2,
                                    BP_Instr1, BP_Instr0)));
                              IF (( BP_ErrSyn4='1' )) THEN
next_BP_ErrSyn4<='1';
                              ELSE next_BP_ErrSyn4<='0';
                              END IF;
                              IF (( BP_ErrSyn1='1' )) THEN
next_BP_ErrSyn1<='1';
                              ELSE next_BP_ErrSyn1<='0';
                              END IF;

                              IF (( BP_DataAccess='1' )) THEN
next_BP_DataAccess<='1';
                              ELSE next_BP_DataAccess<='0';
                              END IF;

                        WHEN ErrSyndSave3 =>
                              next_sreg<=ErrSyndSave4;
                              next_BP_ErrSyn3<='0';
                              next_BP_ErrSyn4<='1';

                              IF (( BP_TRAP='1' )) THEN
next_BP_TRAP<='1';
                              ELSE next_BP_TRAP<='0';
                              END IF;

                              IF (( BP_InstrAccess='1' )) THEN
next_BP_InstrAccess<='1';
                              ELSE next_BP_InstrAccess<='0';
                              END IF;

                              BP_Instr <= ((
std_logic_vector'(BP_Instr23, BP_Instr22, BP_Instr21,
                                    BP_Instr20, BP_Instr19, BP_Instr18,
BP_Instr17, BP_Instr16, BP_Instr15,
                                    BP_Instr14, BP_Instr13, BP_Instr12,
BP_Instr11, BP_Instr10, BP_Instr9,
                                    BP_Instr8, BP_Instr7, BP_Instr6,
BP_Instr5, BP_Instr4, BP_Instr3, BP_Instr2,
                                    BP_Instr1, BP_Instr0)));
                              IF (( BP_ErrSyn2='1' )) THEN
next_BP_ErrSyn2<='1';
                              ELSE next_BP_ErrSyn2<='0';
                              END IF;

                              IF (( BP_ErrSyn1='1' )) THEN
next_BP_ErrSyn1<='1';
                              ELSE next_BP_ErrSyn1<='0';
                              END IF;

                              IF (( BP_DataAccess='1' )) THEN
next_BP_DataAccess<='1';
                              ELSE next_BP_DataAccess<='0';
                              END IF;

                        WHEN ErrSyndSave4 =>
                              next_sreg<=InterrInstr;
                              next_BP_ErrSyn4<='0';
                              next_BP_InstrAccess<='1';
                              next_BP_TRAP<='1';

                              IF (( BP_ErrSyn3='1' )) THEN
next_BP_ErrSyn3<='1';
```

```
                              ELSE next_BP_ErrSyn3<='0';
                              END IF;

                              IF (( BP_ErrSyn2='1' )) THEN
next_BP_ErrSyn2<='1';
                              ELSE next_BP_ErrSyn2<='0';
                              END IF;

                              IF (( BP_ErrSyn1='1' )) THEN
next_BP_ErrSyn1<='1';
                              ELSE next_BP_ErrSyn1<='0';
                              END IF;

                              IF (( BP_DataAccess='1' )) THEN
next_BP_DataAccess<='1';
                              ELSE next_BP_DataAccess<='0';
                              END IF;

                              BP_Instr <=
(std_logic_vector'("0010100000000000000000000"));
                        WHEN InterrInstr =>
                              next_sreg<=ISRData;
                              next_BP_DataAccess<='1';
                              next_BP_InstrAccess<='0';
                              next_BP_TRAP<='0';

                              IF (( BP_ErrSyn4='1' )) THEN
next_BP_ErrSyn4<='1';
                              ELSE next_BP_ErrSyn4<='0';
                              END IF;

                              IF (( BP_ErrSyn3='1' )) THEN
next_BP_ErrSyn3<='1';
                              ELSE next_BP_ErrSyn3<='0';
                              END IF;

                              IF (( BP_ErrSyn2='1' )) THEN
next_BP_ErrSyn2<='1';
                              ELSE next_BP_ErrSyn2<='0';
                              END IF;

                              IF (( BP_ErrSyn1='1' )) THEN
next_BP_ErrSyn1<='1';
                              ELSE next_BP_ErrSyn1<='0';
                              END IF;

                              BP_Instr <=
(std_logic_vector'("1111111111111111111111111"));
                        WHEN ISRData =>
                              IF ( E='1' ) THEN
                                    next_sreg<=ISRErrSyndSave1;
                                    next_BP_ErrSyn1<='1';
                                    next_BP_DataAccess<='0';

                                    IF (( BP_TRAP='1' )) THEN
next_BP_TRAP<='1';
                                    ELSE next_BP_TRAP<='0';
                                    END IF;

                                    IF (( BP_InstrAccess='1' )) THEN
next_BP_InstrAccess<='1';
                                    ELSE next_BP_InstrAccess<='0';
                                    END IF;

                                    BP_Instr <= ((
std_logic_vector'(BP_Instr23, BP_Instr22, BP_Instr21,
                                          BP_Instr20, BP_Instr19,
BP_Instr18, BP_Instr17, BP_Instr16, BP_Instr15,
                                          BP_Instr14, BP_Instr13,
BP_Instr12, BP_Instr11, BP_Instr10, BP_Instr9,
                                          BP_Instr8, BP_Instr7,
BP_Instr6, BP_Instr5, BP_Instr4, BP_Instr3, BP_Instr2,
                                          BP_Instr1, BP_Instr0)));
                                    IF (( BP_ErrSyn4='1' )) THEN
next_BP_ErrSyn4<='1';
                                    ELSE next_BP_ErrSyn4<='0';
                                    END IF;

                                    IF (( BP_ErrSyn3='1' )) THEN
next_BP_ErrSyn3<='1';
                                    ELSE next_BP_ErrSyn3<='0';
                                    END IF;

                                    IF (( BP_ErrSyn2='1' )) THEN
next_BP_ErrSyn2<='1';
                                    ELSE next_BP_ErrSyn2<='0';
                                    END IF;

                              END IF;
                              IF ( RFE='1' AND E='0' ) THEN
                                    next_sreg<=NormInst;
                                    next_BP_InstrAccess<='1';
                                    next_BP_DataAccess<='0';

                                    IF (( BP_TRAP='1' )) THEN
next_BP_TRAP<='1';
                                    ELSE next_BP_TRAP<='0';
                                    END IF;

                                    BP_Instr <= ((
std_logic_vector'(BP_Instr23, BP_Instr22, BP_Instr21,
                                          BP_Instr20, BP_Instr19,
BP_Instr18, BP_Instr17, BP_Instr16, BP_Instr15,
                                          BP_Instr14, BP_Instr13,
BP_Instr12, BP_Instr11, BP_Instr10, BP_Instr9,
                                          BP_Instr8, BP_Instr7,
BP_Instr6, BP_Instr5, BP_Instr4, BP_Instr3, BP_Instr2,
                                          BP_Instr1, BP_Instr0)));
                                    IF (( BP_ErrSyn4='1' )) THEN
next_BP_ErrSyn4<='1';
                                    ELSE next_BP_ErrSyn4<='0';
                                    END IF;
                                    IF (( BP_ErrSyn3='1' )) THEN
next_BP_ErrSyn3<='1';
                                    ELSE next_BP_ErrSyn3<='0';
                                    END IF;

                                    IF (( BP_ErrSyn2='1' )) THEN
next_BP_ErrSyn2<='1';
                                    ELSE next_BP_ErrSyn2<='0';
                                    END IF;

                                    IF (( BP_ErrSyn1='1' )) THEN
next_BP_ErrSyn1<='1';
                                    ELSE next_BP_ErrSyn1<='0';
                                    END IF;

                              END IF;
                              IF ( RFE='0' AND E='0' ) THEN
                                    next_sreg<=ISRInst;
                                    next_BP_ErrSyn4<='0';
                                    next_BP_InstrAccess<='1';
                                    next_BP_DataAccess<='0';

                                    IF (( BP_TRAP='1' )) THEN
next_BP_TRAP<='1';
                                    ELSE next_BP_TRAP<='0';
                                    END IF;

                                    BP_Instr <= ((
std_logic_vector'(BP_Instr23, BP_Instr22, BP_Instr21,
                                          BP_Instr20, BP_Instr19,
BP_Instr18, BP_Instr17, BP_Instr16, BP_Instr15,
                                          BP_Instr14, BP_Instr13,
BP_Instr12, BP_Instr11, BP_Instr10, BP_Instr9,
                                          BP_Instr8, BP_Instr7,
BP_Instr6, BP_Instr5, BP_Instr4, BP_Instr3, BP_Instr2,
                                          BP_Instr1, BP_Instr0)));
                                    IF (( BP_ErrSyn3='1' )) THEN
next_BP_ErrSyn3<='1';
                                    ELSE next_BP_ErrSyn3<='0';
                                    END IF;

                                    IF (( BP_ErrSyn2='1' )) THEN
next_BP_ErrSyn2<='1';
                                    ELSE next_BP_ErrSyn2<='0';
                                    END IF;

                                    IF (( BP_ErrSyn1='1' )) THEN
next_BP_ErrSyn1<='1';
                                    ELSE next_BP_ErrSyn1<='0';
                                    END IF;

                              END IF;
                        WHEN ISRErrSyndSave1 =>
                              next_sreg<=ISRErrSyndSave2;
                              next_BP_ErrSyn1<='0';
                              next_BP_ErrSyn2<='1';
                              IF (( BP_TRAP='1' )) THEN
next_BP_TRAP<='1';
                              ELSE next_BP_TRAP<='0';
                              END IF;

                              IF (( BP_InstrAccess='1' )) THEN
next_BP_InstrAccess<='1';
                              ELSE next_BP_InstrAccess<='0';
                              END IF;

                              BP_Instr <= ((
std_logic_vector'(BP_Instr23, BP_Instr22, BP_Instr21,
                                    BP_Instr20, BP_Instr19, BP_Instr18,
BP_Instr17, BP_Instr16, BP_Instr15,
                                    BP_Instr14, BP_Instr13, BP_Instr12,
BP_Instr11, BP_Instr10, BP_Instr9,
                                    BP_Instr8, BP_Instr7, BP_Instr6,
BP_Instr5, BP_Instr4, BP_Instr3, BP_Instr2,
                                    BP_Instr1, BP_Instr0)));
                              IF (( BP_ErrSyn4='1' )) THEN
next_BP_ErrSyn4<='1';
                              ELSE next_BP_ErrSyn4<='0';
                              END IF;

                              IF (( BP_ErrSyn3='1' )) THEN
next_BP_ErrSyn3<='1';
                              ELSE next_BP_ErrSyn3<='0';
                              END IF;

                              IF (( BP_DataAccess='1' )) THEN
next_BP_DataAccess<='1';
                              ELSE next_BP_DataAccess<='0';
                              END IF;

                        WHEN ISRErrSyndSave2 =>
                              next_sreg<=ISRErrSyndSave3;
                              next_BP_ErrSyn2<='0';
                              next_BP_ErrSyn3<='1';

                              IF (( BP_TRAP='1' )) THEN
next_BP_TRAP<='1';
                              ELSE next_BP_TRAP<='0';
                              END IF;

                              IF (( BP_InstrAccess='1' )) THEN
next_BP_InstrAccess<='1';
                              ELSE next_BP_InstrAccess<='0';
                              END IF;

                              BP_Instr <= ((
std_logic_vector'(BP_Instr23, BP_Instr22, BP_Instr21,
                                    BP_Instr20, BP_Instr19, BP_Instr18,
BP_Instr17, BP_Instr16, BP_Instr15,
                                    BP_Instr14, BP_Instr13, BP_Instr12,
BP_Instr11, BP_Instr10, BP_Instr9,
                                    BP_Instr8, BP_Instr7, BP_Instr6,
BP_Instr5, BP_Instr4, BP_Instr3, BP_Instr2,
                                    BP_Instr1, BP_Instr0)));
                              IF (( BP_ErrSyn4='1' )) THEN
next_BP_ErrSyn4<='1';
                              ELSE next_BP_ErrSyn4<='0';
                              END IF;

                              IF (( BP_ErrSyn1='1' )) THEN
next_BP_ErrSyn1<='1';
                              ELSE next_BP_ErrSyn1<='0';
                              END IF;

                              IF (( BP_DataAccess='1' )) THEN
next_BP_DataAccess<='1';
                              ELSE next_BP_DataAccess<='0';
                              END IF;

                        WHEN ISRErrSyndSave3 =>
                              next_sreg<=ISRErrSyndSave4;
                              next_BP_ErrSyn3<='0';
                              next_BP_ErrSyn4<='1';

                              IF (( BP_TRAP='1' )) THEN
next_BP_TRAP<='1';
                              ELSE next_BP_TRAP<='0';
                              END IF;

                              IF (( BP_InstrAccess='1' )) THEN
next_BP_InstrAccess<='1';
                              ELSE next_BP_InstrAccess<='0';
                              END IF;

                              BP_Instr <= ((
std_logic_vector'(BP_Instr23, BP_Instr22, BP_Instr21,

                                    BP_Instr20, BP_Instr19, BP_Instr18,
BP_Instr17, BP_Instr16, BP_Instr15,
                                    BP_Instr14, BP_Instr13, BP_Instr12,
BP_Instr11, BP_Instr10, BP_Instr9,
                                    BP_Instr8, BP_Instr7, BP_Instr6,
BP_Instr5, BP_Instr4, BP_Instr3, BP_Instr2,
                                    BP_Instr1, BP_Instr0)));
                              IF (( BP_ErrSyn2='1' )) THEN
next_BP_ErrSyn2<='1';
                              ELSE next_BP_ErrSyn2<='0';
                              END IF;

                              IF (( BP_ErrSyn1='1' )) THEN
next_BP_ErrSyn1<='1';
                              ELSE next_BP_ErrSyn1<='0';
                              END IF;

                              IF (( BP_DataAccess='1' )) THEN
next_BP_DataAccess<='1';
                              ELSE next_BP_DataAccess<='0';
                              END IF;

                        WHEN ISRErrSyndSave4 =>
```

```
                              next_sreg<=ISRInst;
                              next_BP_ErrSyn4<='0';
                              next_BP_InstrAccess<='1';
                              next_BP_DataAccess<='0';

                              IF (( BP_TRAP='1' )) THEN
next_BP_TRAP<='1';
                              ELSE next_BP_TRAP<='0';
                              END IF;

                              BP_Instr <= ((
std_logic_vector'(BP_Instr23, BP_Instr22, BP_Instr21,
                                    BP_Instr20, BP_Instr19, BP_Instr18,
BP_Instr17, BP_Instr16, BP_Instr15,
                                    BP_Instr14, BP_Instr13, BP_Instr12,
BP_Instr11, BP_Instr10, BP_Instr9,
                                    BP_Instr8, BP_Instr7, BP_Instr6,
BP_Instr5, BP_Instr4, BP_Instr3, BP_Instr2,
                                    BP_Instr1, BP_Instr0)));
                              IF (( BP_ErrSyn3='1' )) THEN
next_BP_ErrSyn3<='1';
                              ELSE next_BP_ErrSyn3<='0';
                              END IF;

                              IF (( BP_ErrSyn2='1' )) THEN
next_BP_ErrSyn2<='1';
                              ELSE next_BP_ErrSyn2<='0';
                              END IF;

                              IF (( BP_ErrSyn1='1' )) THEN
next_BP_ErrSyn1<='1';
                              ELSE next_BP_ErrSyn1<='0';
                              END IF;

                        WHEN ISRInst =>
                              next_sreg<=ISRData;
                              next_BP_DataAccess<='1';
                              next_BP_InstrAccess<='0';
                              next_BP_TRAP<='0';

                              IF (( BP_ErrSyn4='1' )) THEN
next_BP_ErrSyn4<='1';
                              ELSE next_BP_ErrSyn4<='0';
                              END IF;

                              IF (( BP_ErrSyn3='1' )) THEN
next_BP_ErrSyn3<='1';
                              ELSE next_BP_ErrSyn3<='0';
                              END IF;

                              IF (( BP_ErrSyn2='1' )) THEN
next_BP_ErrSyn2<='1';
                              ELSE next_BP_ErrSyn2<='0';
                              END IF;

                              IF (( BP_ErrSyn1='1' )) THEN
next_BP_ErrSyn1<='1';
```

```
                              ELSE next_BP_ErrSyn1<='0';
                              END IF;

                              BP_Instr <=
(std_logic_vector'("111111111111111111111111"));
                        WHEN NormData =>
                              IF ( E='0' ) THEN
                                    next_sreg<=NormInst;
                                    next_BP_InstrAccess<='1';
                                    next_BP_DataAccess<='0';

                                    IF (( BP_TRAP='1' )) THEN
next_BP_TRAP<='1';
                                    ELSE next_BP_TRAP<='0';
                                    END IF;

                                    BP_Instr <= ((
std_logic_vector'(BP_Instr23, BP_Instr22, BP_Instr21,
                                          BP_Instr20, BP_Instr19,
BP_Instr18, BP_Instr17, BP_Instr16, BP_Instr15,
                                          BP_Instr14, BP_Instr13,
BP_Instr12, BP_Instr11, BP_Instr10, BP_Instr9,
                                          BP_Instr8, BP_Instr7,
BP_Instr6, BP_Instr5, BP_Instr4, BP_Instr3, BP_Instr2,
                                          BP_Instr1, BP_Instr0)));
                                    IF (( BP_ErrSyn4='1' )) THEN
next_BP_ErrSyn4<='1';
                                    ELSE next_BP_ErrSyn4<='0';
                                    END IF;

                                    IF (( BP_ErrSyn3='1' )) THEN
next_BP_ErrSyn3<='1';
                                    ELSE next_BP_ErrSyn3<='0';
                                    END IF;

                                    IF (( BP_ErrSyn2='1' )) THEN
next_BP_ErrSyn2<='1';
                                    ELSE next_BP_ErrSyn2<='0';
                                    END IF;

                                    IF (( BP_ErrSyn1='1' )) THEN
next_BP_ErrSyn1<='1';
                                    ELSE next_BP_ErrSyn1<='0';
                                    END IF;

                              END IF;
                              IF ( E='1' ) THEN
                                    next_sreg<=ErrSyndSave1;
                                    next_BP_DataAccess<='0';
                                    next_BP_ErrSyn1<='1';

                                    IF (( BP_TRAP='1' )) THEN
next_BP_TRAP<='1';
                                    ELSE next_BP_TRAP<='0';
                                    END IF;
                                    IF (( BP_InstrAccess='1' )) THEN
next_BP_InstrAccess<='1';
                                    ELSE next_BP_InstrAccess<='0';
                                    END IF;

                                    BP_Instr <= ((
std_logic_vector'(BP_Instr23, BP_Instr22, BP_Instr21,
                                          BP_Instr20, BP_Instr19,
BP_Instr18, BP_Instr17, BP_Instr16, BP_Instr15,
                                          BP_Instr14, BP_Instr13,
BP_Instr12, BP_Instr11, BP_Instr10, BP_Instr9,
                                          BP_Instr8, BP_Instr7,
BP_Instr6, BP_Instr5, BP_Instr4, BP_Instr3, BP_Instr2,
                                          BP_Instr1, BP_Instr0)));
                                    IF (( BP_ErrSyn4='1' )) THEN
next_BP_ErrSyn4<='1';
                                    ELSE next_BP_ErrSyn4<='0';
                                    END IF;

                                    IF (( BP_ErrSyn3='1' )) THEN
next_BP_ErrSyn3<='1';
                                    ELSE next_BP_ErrSyn3<='0';
                                    END IF;

                                    IF (( BP_ErrSyn2='1' )) THEN
next_BP_ErrSyn2<='1';
                                    ELSE next_BP_ErrSyn2<='0';
                                    END IF;

                              END IF;
                        WHEN NormInst =>
                              next_sreg<=NormData;
                              next_BP_DataAccess<='1';
                              next_BP_InstrAccess<='0';

                              IF (( BP_TRAP='1' )) THEN
next_BP_TRAP<='1';
                              ELSE next_BP_TRAP<='0';
                              END IF;

                              BP_Instr <= ((
std_logic_vector'(BP_Instr23, BP_Instr22, BP_Instr21,
                                    BP_Instr20, BP_Instr19, BP_Instr18,
BP_Instr17, BP_Instr16, BP_Instr15,
                                    BP_Instr14, BP_Instr13, BP_Instr12,
BP_Instr11, BP_Instr10, BP_Instr9,
                                    BP_Instr8, BP_Instr7, BP_Instr6,
BP_Instr5, BP_Instr4, BP_Instr3, BP_Instr2,
                                    BP_Instr1, BP_Instr0)));
                              IF (( BP_ErrSyn4='1' )) THEN
next_BP_ErrSyn4<='1';
                              ELSE next_BP_ErrSyn4<='0';
                              END IF;

                              IF (( BP_ErrSyn3='1' )) THEN
next_BP_ErrSyn3<='1';
                              ELSE next_BP_ErrSyn3<='0';
                              END IF;

                              IF (( BP_ErrSyn2='1' )) THEN
next_BP_ErrSyn2<='1';
                              ELSE next_BP_ErrSyn2<='0';
                              END IF;

                              IF (( BP_ErrSyn1='1' )) THEN
next_BP_ErrSyn1<='1';
                              ELSE next_BP_ErrSyn1<='0';
                              END IF;

                        WHEN OTHERS =>
                  END CASE;

                  next_BP_Instr23 <= BP_Instr(23);
                  next_BP_Instr22 <= BP_Instr(22);
                  next_BP_Instr21 <= BP_Instr(21);
                  next_BP_Instr20 <= BP_Instr(20);
                  next_BP_Instr19 <= BP_Instr(19);
                  next_BP_Instr18 <= BP_Instr(18);
                  next_BP_Instr17 <= BP_Instr(17);
                  next_BP_Instr16 <= BP_Instr(16);
                  next_BP_Instr15 <= BP_Instr(15);
                  next_BP_Instr14 <= BP_Instr(14);
                  next_BP_Instr13 <= BP_Instr(13);
                  next_BP_Instr12 <= BP_Instr(12);
                  next_BP_Instr11 <= BP_Instr(11);
                  next_BP_Instr10 <= BP_Instr(10);
                  next_BP_Instr9 <= BP_Instr(9);
                  next_BP_Instr8 <= BP_Instr(8);
                  next_BP_Instr7 <= BP_Instr(7);
                  next_BP_Instr6 <= BP_Instr(6);
                  next_BP_Instr5 <= BP_Instr(5);
                  next_BP_Instr4 <= BP_Instr(4);
                  next_BP_Instr3 <= BP_Instr(3);
                  next_BP_Instr2 <= BP_Instr(2);
                  next_BP_Instr1 <= BP_Instr(1);
                  next_BP_Instr0 <= BP_Instr(0);
            END PROCESS;

            PROCESS (BP_DataAccess)
            BEGIN
                  IF (( BP_DataAccess='1' )) THEN DataAccess<='1';
                  ELSE DataAccess<='0';
                  END IF;
            END PROCESS;

            PROCESS (BP_ErrSyn1)
            BEGIN
                  IF (( BP_ErrSyn1='1' )) THEN ErrSyn1<='1';
                  ELSE ErrSyn1<='0';
                  END IF;
            END PROCESS;

            PROCESS (BP_ErrSyn2)
            BEGIN
```

```
                IF (( BP_ErrSyn2='1' )) THEN ErrSyn2<='1';
                ELSE ErrSyn2<='0';
                END IF;
          END PROCESS;

          PROCESS (BP_ErrSyn3)
          BEGIN
                IF (( BP_ErrSyn3='1' )) THEN ErrSyn3<='1';
                ELSE ErrSyn3<='0';
                END IF;
          END PROCESS;

          PROCESS (BP_ErrSyn4)
          BEGIN
                IF (( BP_ErrSyn4='1' )) THEN ErrSyn4<='1';
                ELSE ErrSyn4<='0';
                END IF;
          END PROCESS;

          PROCESS (BP_InstrAccess)
          BEGIN
                IF (( BP_InstrAccess='1' )) THEN InstrAccess<='1';
                ELSE InstrAccess<='0';
                END IF;
          END PROCESS;

          PROCESS (BP_TRAP)
          BEGIN
                IF (( BP_TRAP='1' )) THEN TRAP<='1';
                ELSE TRAP<='0';
                END IF;
          END PROCESS;

          PROCESS
(BP_Instr0,BP_Instr1,BP_Instr2,BP_Instr3,BP_Instr4,BP_Instr5,

      BP_Instr6,BP_Instr7,BP_Instr8,BP_Instr9,BP_Instr10,BP_Instr11,BP_
Instr12,

      BP_Instr13,BP_Instr14,BP_Instr15,BP_Instr16,BP_Instr17,BP_Instr18
,BP_Instr19,
                BP_Instr20,BP_Instr21,BP_Instr22,BP_Instr23,Instr)
          BEGIN
                Instr <= (( std_logic_vector'(BP_Instr23, BP_Instr22,
BP_Instr21,
                      BP_Instr20, BP_Instr19, BP_Instr18, BP_Instr17,
BP_Instr16, BP_Instr15,
                      BP_Instr14, BP_Instr13, BP_Instr12, BP_Instr11,
BP_Instr10, BP_Instr9,
                      BP_Instr8, BP_Instr7, BP_Instr6, BP_Instr5,
BP_Instr4, BP_Instr3, BP_Instr2,
                      BP_Instr1, BP_Instr0)));
                Instr0 <= Instr(0);
                Instr1 <= Instr(1);
                Instr2 <= Instr(2);
                Instr3 <= Instr(3);
                Instr4 <= Instr(4);
                Instr5 <= Instr(5);
                Instr6 <= Instr(6);
                Instr7 <= Instr(7);
                Instr8 <= Instr(8);
                Instr9 <= Instr(9);
                Instr10 <= Instr(10);
                Instr11 <= Instr(11);
                Instr12 <= Instr(12);
                Instr13 <= Instr(13);
                Instr14 <= Instr(14);
                Instr15 <= Instr(15);
                Instr16 <= Instr(16);
                Instr17 <= Instr(17);
                Instr18 <= Instr(18);
                Instr19 <= Instr(19);
                Instr20 <= Instr(20);
                Instr21 <= Instr(21);
                Instr22 <= Instr(22);
                Instr23 <= Instr(23);
          END PROCESS;
      END BEHAVIOR;

      LIBRARY ieee;
      USE ieee.std_logic_1164.all;

      ENTITY INTERRUP IS
            PORT (Instr : OUT std_logic_vector (23 DOWNTO 0);
                  CLK,E,RESET,RFE: IN std_logic;

      DataAccess,ErrSyn1,ErrSyn2,ErrSyn3,ErrSyn4,InstrAccess,TRAP : OUT
std_logic
                        );
      END;

      ARCHITECTURE BEHAVIOR OF INTERRUP IS
            COMPONENT SHELL_INTERRUP
                  PORT (CLK,E,RESET,RFE: IN std_logic;

      DataAccess,ErrSyn1,ErrSyn2,ErrSyn3,ErrSyn4,Instr0,Instr1,Instr2,I
nstr3,

      Instr4,Instr5,Instr6,Instr7,Instr8,Instr9,Instr10,Instr11,Instr12
,Instr13,

      Instr14,Instr15,Instr16,Instr17,Instr18,Instr19,Instr20,Instr21,I
nstr22,
                              Instr23,InstrAccess,TRAP : OUT
std_logic);
            END COMPONENT;
      BEGIN
            SHELL1_INTERRUP : SHELL_INTERRUP PORT MAP
(CLK=>CLK,E=>E,RESET=>RESET,RFE=>

      RFE,DataAccess=>DataAccess,ErrSyn1=>ErrSyn1,ErrSyn2=>ErrSyn2,ErrS
yn3=>ErrSyn3
```

```
      ,ErrSyn4=>ErrSyn4,Instr0=>Instr(0),Instr1=>Instr(1),Instr2=>Instr
(2),Instr3=>

      Instr(3),Instr4=>Instr(4),Instr5=>Instr(5),Instr6=>Instr(6),Instr
7=>Instr(7),

      Instr8=>Instr(8),Instr9=>Instr(9),Instr10=>Instr(10),Instr11=>Ins
tr(11),

      Instr12=>Instr(12),Instr13=>Instr(13),Instr14=>Instr(14),Instr15=
>Instr(15),

      Instr16=>Instr(16),Instr17=>Instr(17),Instr18=>Instr(18),Instr19=
>Instr(19),

      Instr20=>Instr(20),Instr21=>Instr(21),Instr22=>Instr(22),Instr23=
>Instr(23),
                  InstrAccess=>InstrAccess,TRAP=>TRAP);
      END BEHAVIOR;
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B: SPACE EXPERIMENT REVIEW BOARD

The original intent for the CFTP was to be a part of the Naval Postgraduate School Satellite NPSAT-1. After talking with the Space Test Program Office, it was decided that the CFTP should request additional flights in order to increase the test data available for the design. This required an extensive amount of time and research into the process whereby space experiments are approved by the Department of Defense for space flights. This Appendix will go into detail on this process, called the Space Experiment Review Board (SERB).

A. SERB OVERVIEW

The SERB process has two stages. A Service Board, presided over in the case of CFTP by the Navy, and a Department Board. Both boards have the same criteria: military relevance, quality of experiment and service priority.

1. Military Relevance

Military relevance is 60% of the overall grade for an experiment. It is intended to ensure that the experiment does pertain directly to the military. While science experiments are allowed, the goal is to apply the experiment results to the war fighter in particular [9].

2. Quality of Experiment

The quality of the experiment is 20% of the overall grade, and it is intended to ensure that, all other factors being equal, experiments that are only in a conceptual phase do not get too heavy a precedence over experiments that are largely completed.

3. Service Priority

Service Priority takes into account the previous two criteria and is a numerical ranking of the experiment against the entire group of experiments being presented that year. In the example of the CFTP, it was ranked 13 of 24 experiments at the Navy SERB in 2002.

Service Priority serves two different purposes at a Service Board. If the experiment has been presented in the past, it is an indication to the current board members of the experiment's status the previous year. It is also used to prioritize experiments at the Department Board for the Service experiments. Again using the CFTP as an example, its ranking of 13 of 24 placed it in the middle of the Navy's experiments. This gave the Department Board members an indication of the Navy's priority for the experiment.

B. SERB DOCUMENTATION

The documentation required for the SERB process is not extensive, but it is quite critical to the process. There are two documents which the SERB requires, the Space Test Program Flight Request (DD Form 1721) and the Space Test Program Flight Request Executive Summary (DD Form 1721/1) [9].

1. Space Test Program Flight Request

DD Form 1721 contains all the information on the experiment. It includes the objective of the experiment, experiment dimensions and mass, launch platform preferences, orbital parameters, and communication requirements.

2. Space Test Program Flight Request Executive Summary

The Executive Summary contains the more important information from the DD 1721. It includes the points of contact for the experiment, the dimensions, funding data, orbital parameters, and launch platform data.

C. PRESENTATION

Each experiment team is given ten minutes to brief the experiment. A Power Point Presentation template of five slides is provided to the team. This template has the minimum information required for presentation. It includes the concept, justification, a detailed overview, summary of data application and flight mode suitability [9].

1. Concept

The concept is intended to provide the intent of the experiment. It should include a description of the experiment, including performance parameters, and a graphical representation of the experiment. Figure 28 shows the graphical representation used by the CFTP at the Department Board in 2002.

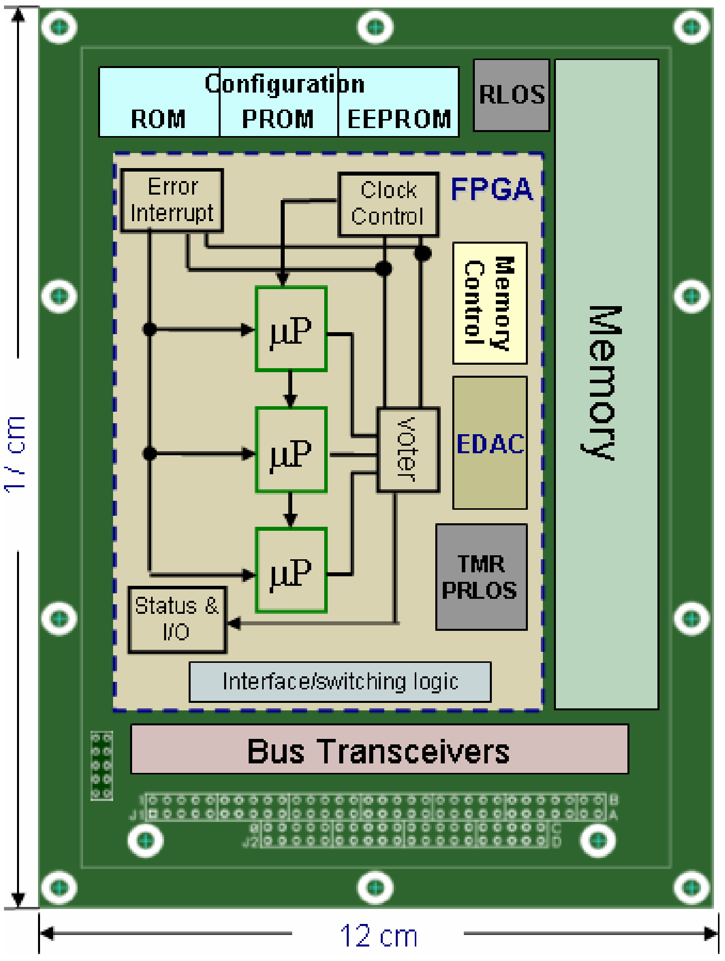


Figure 28. SERB Graphical Representation

2. Justification

Justification is perhaps the most critical portion of the presentation. With this portion of the presentation, the presenter is using various documents and requirements of the Department of Defense to justify the experiment. The documents include the US Space Command Long Range Plan, Defense Technology Area Plan, and the Air Force Space Command Strategic Master Plan.

In these documents are specific requirements for war fighting support capability that can be accomplished with space assets. Using this information, the experiment presenter can show that the experiment is capable of satisfying DoD requirements.

3. Detailed Overview

The detailed overview lists the flight data parameters of the experiment. This includes the types of orbit desired, such as geosynchronous or low earth orbit. The weight and size of the experiment are also listed here. The priority ranking given during the last SERB cycle and the services requested from the Space Test Program Office are next. Finally comes the funding required versus actual funding broken down year-by-year through the life of the experiment.

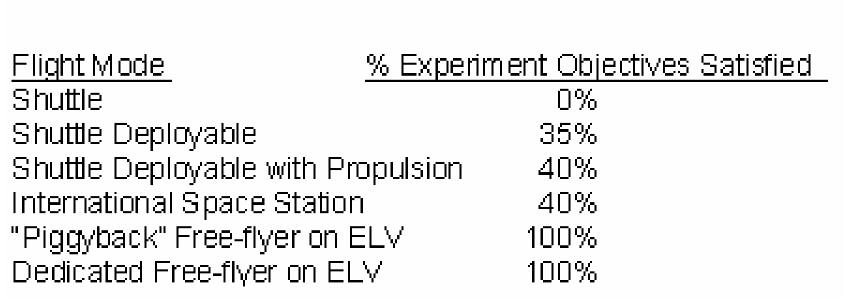
4. Summary of Data Application

The summary of data application is an explanation of what the data the experiment gathers will be used for. It must include the category of research in which the experiment is. In the case of the CFTP, it is applied research.

5. Flight Mode Suitability

The final slide lists the different types of platforms the experiment can ride on. These include the Space Shuttle, deployed from the Shuttle, deployed from the Shuttle with propulsion, the International Space Station, Piggyback on a Free-Flyer, or Dedicated Free-Flyer. A Free-Flyer is a satellite such as a GPS or communications satellite. A Piggyback flight is one where the experiment is not the primary payload of the satellite.

The experiment team must list each of these flight modes and their ability to meet the experiment objectives. In the case of the CFTP the Shuttle was unsatisfactory, since it is such a short duration flight. Table 4 shows the full Flight Mode Suitability table for the CFTP from the 2002 SERB.



Flight Mode	% Experiment Objectives Satisfied
Shuttle	0%
Shuttle Deployable	35%
Shuttle Deployable with Propulsion	40%
International Space Station	40%
"Piggyback" Free-flyer on ELV	100%
Dedicated Free-flyer on ELV	100%

Table 4. Flight Mode Suitability for the CFTP

D. EXPERIMENT MANIFESTATION

Once the DoD SERB is complete, the Space Test Program Office (STP) reviews the list of experiments and determines flight manifestation. This is done in the manner that is most efficient to the DoD, not by a direct ranking priority.

For example, STP may have a satellite with space available for a small experiment. If the highest ranked experiment from the SERB is too large to get on this particular satellite, the second ranked experiment (or even further down the list) may actually be manifested first.

This was the case for the CFTP. Two satellites had space available for a small experiment. Since the CFTP is small, light weight, and low in power, it is much easier to find flights for than larger experiments.

The CFTP requested four flights during the 2002 DoD SERB: three at different inclinations in Low Earth Orbit (LEO) and one in a highly elliptical orbit such as a Geosynchronous Transfer Orbit (GTO).

The purpose of the four orbits is to test the fault tolerance capability in both benign and high radiation environments. In LEO, it expected that only a few SEUs will occur in a week. With these orbits, testing basic functionality will be possible without concern for frequent SEU interrupts. If the design is faulty, infrequent SEUs will allow for testing reconfiguration until a more robust design can be implemented.

The GTO will test for severe radiation tolerance. In a GTO, the CFTP will pass through the Van Allen radiation belts. In this environment, SEU frequency is expected to be significantly higher. If the CFTP can function properly in this high radiation environment, it is expected that it can function in any radiation environment.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Lashomb, Peter A., "Triple Modular Redundant (TMR) Microprocessor System for Field Programmable Gate Array (FPGA) Implementation", Master's Thesis, Naval Postgraduate School, Monterey, California, March 2002.

2. Clark, Kenneth A., *The Effect of Single Event Transients on Complex Digital Systems,* Doctoral Dissertation, Naval Postgraduate School, Monterey, California, June 2002.

3. Payne, John C., Fault Tolerant Computing Testbed: A Tool for the Analysis of Hardware and Software Fault Handling Techniques", Master's Thesis, Naval Postgraduate School, Monterey, California, December 1998.

4. Hennessy, John L. and Patterson, David A., *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, 1990.

5. *Xilinx Programmable Logic Data Book*, Xilinx Inc., San Jose, California, 1999.

6. "Virtex-II Platform FPGAs: Introduction and Overview" http://direct.xilinx.com/bvdocs/publications/ds031.pdf, February 2003.

7. Email from Timothy Martin of Aero Corp to Nathan Beltz of NPS, September, 2002.

8. *Xilinx ISE 5 Libraries Guide*, Xilinx Inc., San Jose, California, 2002.

9. "Space Test Program Office Homepage" http://spacescience.nrl.navy.mil/stp/index.html, March 2003.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Herschel H. Loomis
   Naval Postgraduate School
   Monterey, California

4. Professor Alan A. Ross
   Naval Postgraduate School
   Monterey, California

5. Doctor Kenneth A. Clark
   Naval Research Laboratory
   Washington, DC

6. LCDR Joe Reason, USN
   National Reconnaissance Office
   Chantilly, Virginia

7. Capt. Brian Bailey, USAF
   National Reconnaissance Office
   Chantilly, Virginia

8. LT Paula Travis, USN
   Space Test Program Office, Det 12
   Albuquerque, New Mexico